HEWLETT PACKARD

HP 9000
Computers

A Beginner's Guide to HP-UX

A Beginner's Guide to HP-UX

HP 9000 Computers

HP Part No. B1862-90000
Printed in England January 1991

First Edition
E0191

Legal Notices

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty. A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updates or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

January 1991 ... Edition 1. First combined edition of Beginner's Guides, including `vi`, Shells, and HP-UX. This edition applies to HP 9000, Series 300, 400, 600, 700, 800 computers.

HP-UX Quick Reference

How to use this card:

1. To refer to selected HP-UX commands, place this card near your display.
2. Type the commands as they are shown in the second column below.
3. Include paths with file names, if you are working with different directories.
4. Follow each command with [Return].

To Do This ...	Type This ...
Working with Directories:	
Show current working directory	`pwd`
Change directory	`cd` *directory_path*
Change to home directory	`cd`
Create a directory	`mkdir` *directory_name*
Remove an (empty) directory	`rmdir` *directory_name*
Display permissions for a directory	`ll -d` *directory_name*
Change permissions for file or directory	`chmod` *permission_nos name*
Symbolically change file or directory permissions	`chmod` *class=permissions name*
Working with Files:	
Read mail	`elm`
List files and directories in current directory	`ls`
List all files or directories, including invisible ("dot") files	`ls -a`
List files, and show directories with "/"	`lsf`
Display permissions for a file	`ll` *file_name*
Create or edit a file	`vi` *file_name*
Display file contents	`more` *file_name* ([q] to quit)
Copy a file	`cp` *file_name file_copy*
Move a file to a new file name	`mv` *old_file new_file*
Append *file1* onto the end of *file2*	`cat` *file1* >> *file2*
Remove *file*	`rm` *file*
Display the first 10 lines of a file	`head` *file_name*
Check the spelling in a file	`spell` *file_name*
Print a file	`lp` *file_name*

HEWLETT PACKARD

To Do This ...	Type This ...
Finding and Organizing:	
Find file(s) beginning with *x* in current- and sub-directories	`find . -name 'x*' -print`
Find all occurrences of *word* in all files in current directory	`grep` *word* `*`
Sort *listfile* of two-word names by last name	`sort +1` *listfile*
Display date and time	`date`
List all aliases	`alias`
Find HP-UX command information	`man` *command_name*
Determine PATH setting	echo $PATH
System Operations:	
Clear screen	`clear`
Set command-line editor	`set -o` *editor_name*
Edit your command line (in Korn/Key Shell set for `vi`)	ESC (use `vi` commands)
Recall previous command line (with `vi` editor)	ESC `k` (back) or `j` (forward)
Execute previous command line	Return (when line is displayed.)
Set terminal type (select *term_type* from `/usr/lib/terminfo`)	TERM=*term_type*
List current process status and *PID's*	`ps -ef`
Kill (terminate) a process	`kill` *PID*
Create or change a password	`passwd`
Redirect input from a file to a command	*command* `<` *infile*
Connect two processes with a "pipe"	*command1* `|` *command2*
Network Operations:	
Invoke `ftp` and connect to *remote_host*	`ftp` *remote_host*
Copy files from *remote_host* with `ftp`	`get` *remote_file* (while in `ftp`)
Copy files from local current directory to current directory on *remote_host*	`put` *local_file* (while in `ftp`)
Exit `ftp`	`bye`
Copy a file from your *local_file* to a file on a remote host, using `rcp`	`rcp` *local_file* *remote_hostname:remote_file*
Copy a file from a remote host to your local directory, using `rcp`	`rcp` *remote_hostname:* *remote_file local_file*
Log in on a remote system	`rlogin` *remote_hostname*

vi Editor Quick Reference

How to use this card:

1. To refer to selected `vi` commands, place this card near your display.
2. Type the commands as they are shown in the second column below.
3. Follow colon (`:`) and search (`/`) commands with Return, as indicated.

To Do This ...	Type This in Command Mode
Getting Started with vi:	
Read, create, or edit a file (*enter from shell*)	$: `vi` *filename*
Make sure you're in command mode	ESC
Show whether you are in command or insert mode	`:set showmode` Return
Find out what file you are in	CTRL-g
Moving Within the File:	
Move the cursor left or right	Use `h` or `l` key.
Move the cursor up or down	Use `k` or `j` key.
Cursor to end of line	`$`
Cursor to beginning of line	`0` (zero)
Cursor to end of file	`G`
Cursor to beginning of file	`1G` (one G)
Editing Text: Inserting:	
Insert text at the cursor position	`i`
Insert (append) text *after* the cursor position	`a`
Backspace to overwrite previous character (*in Insert Mode*)	CTRL-h or Back space
Editing Text: Deleting:	
Delete characters at cursor	`x`
Delete word (cursor at beginning)	`dw`
Delete entire line at cursor position	`dd`
Delete from cursor to end of current line	`D`
Delete from cursor to end of file	`dG`
Moving Text:	
Join lines of text	`J`
Copy a line into a buffer	`yy`
Put copied or deleted text line after cursor line	`p`
Read in another file after cursor line	`:r` *filename* Return

HEWLETT PACKARD

To Do This ...	Type This in Command Mode
Searching and Marking Text:	
Search for *words*	*/words* Return
Repeat the previous search for *words*	`n` (forward) or `N` (backward)
Place invisible mark at current cursor position	`m`(*a-z*)
Move cursor to invisible marker	:(*'a-'z*) Return
Saving and Printing Files:	
Save file in same file name	`:w` Return
Save file to a new file name	`:w` *new_file_name* Return
Exit from file without saving changes you've made	`:q!`
Overwrite another existing file with this file	`:w!` *other_file_name* Return
Save and exit from file	`:wq` Return
Print a file from within that same file	`:!lp %` Return
Print another file from within a file	`:!lp` *file_name* Return
Repairing Mistakes:	
"Undo" the previous action	`u`
Restore a line to its previous state	`U`
Restore ("put") last delete	`p`
Restore current file, disregarding changes since saving	`:e!` Return
Compare changes "before and after" an action	`u` (*repeat to "toggle"*)
Recover a file after a system interruption (from shell)	$: `vi -r` *filename*
In General:	
Execute shell command *cmd* from within file	`:!` *cmd* Return
Set line length (wrapmargin) *n* lines from right	`:set wm=`*n* Return
Display current options	`:set` Return
Display all options	`:set all` Return
Abbreviate *string* with *str*	`:ab` *str string* Return
Display all abbreviations	`:ab` Return
Unabbreviate *str*	`:una` *str* Return

For more information, see *The Ultimate Guide to the vi and ex Text Editors.*

Contents

1

Getting Started

HP-UX is a powerful and versatile system that meets the computing needs of diverse groups of users. You can use HP-UX simply to run applications, or you can develop your own applications in its rich software-development environment. In addition, HP-UX offers powerful subsystems, such as electronic mail, windows, networking, and graphics.

This book introduces you to HP-UX so that you can become productive as soon as possible.

Finding Information in this Manual

The following aids will help you in using your HP-UX system:

- Two indexes:
  - Task Index, giving references to specific task information.
  - Main Index, giving references to commands and general references.
- In the front of this manual, you will find:
  - The *HP-UX Quick Reference*
  - The `vi` *Editor Quick Reference*

  These are tear-out cards which you can place near your system to help you recall the most useful commands while you are learning HP-UX and `vi`.

Before You Begin . . .

This guide was designed to get you "up and running" on HP-UX as quickly as possible. For example, after reading this chapter and Chapter 2, you should immediately be able to start using the system and running some commands.

This chapter leads you through your first work session on HP-UX. You should read and do the specific tasks in this chapter in one sitting and in the order in which they appear.

Before you start this chapter, get your **username** and **password** from the system administrator or the support person who installed your system. You will need this information in the first section, "Logging In."

Chapter Contents

This chapter discusses the following topics:

- How to use this guide.
- Getting to know your system.
- Logging in—identifying yourself to HP-UX so you can start a work session.
- Using the command line.
- Setting your password—to help ensure the security of your system.
- Logging out—exiting from HP-UX when you're finished using the system.

Conventions

This book uses the following typographical conventions within examples:

If you see ...	It means ...
colored text	You type the text exactly as shown. For example, `more sample_file` Return means you should type exactly those characters.
`computer text`	Text displayed by the computer system. For example, `login:` indicates a login prompt displayed by the system.
italic text	You supply the text. For example, `more` *file_name* means that you type **more** followed by a file name of your choice. Italic text also is used as annotation.
▢	You type the corresponding key on the keyboard. For example, CTRL-D means you hold down the CTRL key, and press the D key.

For More Information ...

For details on all the manuals you can get with HP-UX, see *Finding HP-UX Information*.

Notes:

Getting to Know Your System

Throughout this guide, you will see the term **system administrator**. The system administrator is someone who manages your system, taking care of such tasks as adding peripheral devices, adding new users, and doing system backups. In general, this person (who may also be called the **system operator** or something similar) is the one to go to with questions about implementing your software.

However, if you are the only user on a single-user system, then whenever this guide refers you to the system administrator, you should be able to get help from the system administration manuals that you have purchased with your system. If you need other kinds of information, it will help to identify someone in your organization who is experienced in HP-UX and who can provide help with user problems. If such a person isn't available, your HP support engineer can also provide installation and maintenance help, in accordance with your support contract.

Visual Help for the User

Most of this guide assumes you will be using the Bourne, Korn, or Key command interpreters (or **shells** as they are called in HP-UX terminology). If you need to find about using the C Shell, see the Appendix. With these shells, you run a command by typing the command's name on a *command line* and pressing the Return key.

For example, you can run the `date` command by typing:

```
$ date Return

Mon Nov 12 09:54:23 MST 1990
```

Typically, in order to get started with HP-UX, you'll need to refer to the first chapters of this manual for command names and syntax, or you can use the *HP-UX Quick Reference*, which you'll find in the front of the manual, along with the `vi` *Editor Quick Reference*. You can tear these out and place them near your display.

But, if you'd like to get visual help directly from the screen while you are working with commands, the following sections describe some alternatives which HP-UX provides for you.

PAM—The Personal Application Manager

As an alternative to command-line oriented shells, you can opt to use PAM, the Personal Applications Manager. PAM implements a visual, menu-oriented interface that uses arrow keys and the softkeys (f1 ... f8).

To learn more about PAM, refer to the *Shells: User's Guide* and the *pam* entry in section 1 of the *HP-UX Reference*.

The Key Shell: Visual Help for the Korn Shell

The Key Shell gives you help on most HP-UX commands by displaying softkey command-names and options in sequence. You can select from these and let the Key Shell build your command lines "in English" before you have mastered the commands and command syntax of HP-UX. In general, the discussion of the Korn Shell in this manual also applies to the Key Shell. For specific information on using the Key Shell, see Chapter 4, in this manual, and the *Shells: User's Guide*,

TSM: Window-Like Functions for Terminal Users

TSM ("Terminal Session Manager") allows you to run multiple HP-UX programs, each in its own session, on your terminal. TSM provides an easy method for switching quickly between these sessions making your terminal appear to be ten "terminals." Each of these "terminals", called a TSM session, can run its own independent program. Thus, TSM provides some of the functionality of a windowing system like VUE or X-Windows.

When TSM starts, it will automatically start a list of programs that you specify.

TSM has two operating modes: menu mode and command mode. In menu mode, you can quickly select the desired task from a pull-down menu. Command mode allows you to "flip" back and forth between the programs in each session as if you were moving your eyes and fingers from one terminal to another.

For more information on TSM, see the *Terminal Session Manager: User's Guide* and the *tsm* entry in section 1 of the *HP-UX Reference*.

VUE: the Visual User Environment

VUE is a graphical user interface based on the X Window System and OSF/Motif standards. It has several components which will help you use your system faster and more intuitively:

- Login Manager. This gives you a window to enter your name and password at login, and makes it easier to correct mistakes in doing so. You can also specify which language you want HP VUE to use in the Login Manager.
- File Manager. By manipulating icons, this interface allows you to navigate through directories on your local system and on remote workstations.
- Workspace Manager. This lets you manage and navigate through multiple workspaces (logical collections of windows and icons). You can create and use multiple workspaces for different tasks.
- The Style Manager. You can customize your working environment and startup configuration through the use of this system aid.
- Help Manager. This gives you easy access to man pages containing information on HP-UX commands, and to other system information.

For More Information on VUE

The following manuals will give you further guidance in using VUE:

- *HP Visual User Environment User's Guide.*
- *HP Visual User Environment System Administration Manual.*

Your Shell Command Interpreter

Whenever you enter a command, you are making use of a command interpreter which interprets that command for HP-UX. The command interpreter is called a **shell**.

When you log in, you are said to be "in" a **shell**. HP-UX supports several different shells which behave in slightly different ways and give you varying amounts of interactive support at your display. These are called the Bourne, Korn, Key, C, and PAM Shells. Your system administrator determines which shell you get when you first log in, and you have the option of changing shells later.

If, after logging in, you see a Personal Applications Manager menu screen, your shell is the PAM Shell. Otherwise, you can determine which shell you're "in" by typing the following command:

```
$ echo $SHELL
```

If `echo $SHELL` displays ...	Then your shell is ...
`/bin/sh`	Bourne Shell
`/bin/csh`	C Shell
`/bin/ksh`	Korn Shell
`/usr/bin/keysh`	Key Shell

You'll find more details about this, as well as information about the features of each of these shells in Chapter 9 and Chapter 10 in this manual.

For More Information on Shells

Your shell has many productivity-enhancing capabilities you may find useful after you've mastered the basics in this book. To learn about shells in detail, also refer to *Shells: User's Guide*.

Logging In

To begin using HP-UX, you must log in. When you log in, HP-UX prompts you for your **username** and **password** (if you have one). On some systems, you may also have to respond to a "`TERM = (hp)`" prompt. When you have finished logging in, you should see a command line prompt which means that you can proceed with entering commands on the system.

If you are using VUE (Visual User Environment) or the X Window System, you have an automatic login window which will provide a form for your entries, once your login and password are set.

1. After you boot your system, you should see this prompt:

   ```
   login:
   ```

   If you don't see this prompt, consult your system administrator or *HP-UX System Administration Tasks*. If you are working with a system equipped with NLIO, see your *NLIO System Administration* manual.

2. When you see the login prompt, type your username and press Return. Your username identifies you as a valid user of the system. HP-UX lets you log in only if you have a username. For example, if your username were `leslie`, you would type:

   ```
   login: leslie
   ```
   Return

3. Unless you are using VUE, the Back space key does not work during login. If you make a mistake when typing, press Return once or twice and HP-UX will display a new login prompt, so you can re-enter your username.

4. If you have a password, you should see a password prompt after entering your username:

   ```
   Password:
   ```

5. Your password is an "invisible" codeword known only to you. Passwords help ensure your system's security. When you see the password prompt, type your password and press Return. To ensure that other users cannot see your password, HP-UX does *not* display the characters as you type them. If you do *not* have a password, see your system administrator or skip to step 6. To see how to create a new password on your system, see the section, "Setting Your Password", in this chapter.

6. If you are using VUE, you will be using a special log-in window rather than using the command-line procedure. In the password section of the window, your "invisible" password will be indicated by a series of asterisks as you type the characters. You may backspace over these characters if you need to retype some or all of your password.

7. Every terminal or graphics display has a *terminal type* (or *term-type*), which HP-UX must know to communicate with the terminal. If HP-UX doesn't set the term-type automatically, you will be prompted for it: `TERM = (hp)`

   If you see this prompt, ask your system administrator for your term-type, and enter it. (You'll also find a listing of possible settings in Table 9-7 in Chapter 9). For example, if you use a "2392a" terminal, type:

   ```
   TERM = (hp) 2392a Return
   ```

   If you *don't* know your term-type, enter nothing and press Return. The system will automatically set up your display according to default values appropriate to most HP terminals.

Note If your display should behave strangely, especially when you are working in `vi`, the cause will likely be in your term-type setting. If this happens, check, and, if necessary reset your TERM variable.

---

8. If you have completed the preceding steps correctly, HP-UX typically displays a welcome and copyright message, followed by a **command prompt**. A command prompt indicates the system is ready to accept commands. Typical prompts are **$** or **%**.
9. If you are using PAM (Personal Applications Manager), you'll see the PAM menu instead of a command prompt. If you see a prompt, you can proceed to the next section "Typing Commands."

If Problems Occur . . .

Listed below are some messages you might see during login and what to do if you see them. (If you see other messages, consult your system administrator.)

If you see the message . . .	Do this . . .
`Invalid login.` `login:`	Check to see if you have made a mistake when typing either your username or password. Then try again.

`Your password has expired. Choose a new one.`
`Changing password for leslie`
`New password:`
	Enter a new password. See the section "Setting Your Password", in this chapter.

`Maximum number of users already logged in.`	You must wait until someone logs out before you can log in. Try again later.

Using the Command Line

To run a command, type the command's name after the command line prompt, and press Return. To correct typing mistakes, use the Back space key.

The Command Line Prompt

When you see the command line prompt, you can begin typing commands. By default, the command line prompt is either **$** or **%**, but it can be different, depending on how your system administrator has set up your account. In any case, you can locate the prompt by pressing Return several times; HP-UX displays the prompt every time you press Return:

```
$ Return
$ Return
$
```

For the purposes of clarity and consistency, examples throughout this book use the **$** *prompt.*

Running Commands

To run a command, type the command's name after the prompt and press Return. The command then will begin running. When the command finishes, the prompt reappears. For example, run the following **whoami** command now:

```
$ whoami Return
leslie          Your user name appears here.
$               Then the command line prompt reappears.
```

If you make a mistake when typing a command, use the Back space key to back up and correct it.

(To make examples as clear as possible, the remainder of this book will *not* show the Return key at the end of each command line.)

To get the feel of using commands, run the `date` and `cal` commands:

```
$ date
Mon Sep 24 16:56:07 MDT 1990
```

Displays the current date and time.

```
$ cal

   September 1990
 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Displays an English calendar for the current month.

You'll get more practice at using the basic commands of HP-UX as you go through the following chapters in this manual.

Setting Your Password

Passwords help prevent unauthorized users from logging in to the system. If you don't have a password, set one that only you know. When you've set a password, change it occasionally to ensure system security. Use the `passwd` command to set or change your password. (To learn more about system security and selecting a secure password, refer to Chapter 12 "Making Your System Secure.")

Choosing a New Password

Choose your new password according to these rules:

- The password must contain between six and eight characters.
- At least two characters must be letters (uppercase or lowercase).
- At least one character must be either of the following:
  - Numeric (the digits 0 through 9).
  - Special (neither letters nor numeric—for example, -, _, or $).

According to these rules, the following are all valid passwords:

```
foo-bar     $money$     Number_9     @rophy
```

Also, uppercase and lowercase letters are different. Thus, these are all different passwords:

```
foo-bar     Foo-Bar     FOO-BAR
```

When you choose a password, you want to ensure that no one can guess what you chose. If someone knows your password, that person may log in and access your files. Chapter 12 "Making Your System Secure" offers suggestions on how to select and protect your password. These guidelines are of particular significance if you work with sensitive material.

Running the passwd Command

When you've chosen your new password, set it by running `passwd`:

```
$ passwd
```

After you've entered the `passwd` command, it leads you through these steps:

1. If you don't have a password, skip this step. If you *do* have a password, `passwd` prompts you to enter it.

   ```
   Changing password for leslie    Your username appears here.
   Old password:                   Enter your current password.
   ```

2. When you see the following prompt, enter your new password.

   ```
   New password:
   ```

3. Next, you need to re-enter the new password to verify it:

   ```
   Re-enter your new password:
   ```

Now, when you log in again, use your new password.

If Problems Occur ...

If you see this ...	You might have done this ...
`Sorry.`	Typed your current password incorrectly to the "`Old password:`" prompt. Run `passwd` again.
`Password is too short - must be at least 6 characters.`	Specified a new password with less than six characters.
`Password must contain at least two alphabetic characters and at least one numeric or special character.`	Specified a new password that doesn't conform to the rules described earlier.
`They don't match; try again.`	Typed your new password differently when prompted to re-enter it.
`Too many failures - try later.`	Made too many mistakes. Try again.

Logging Out

When you're finished with your HP-UX work session, log out using the `exit` command. On some systems, you can use (CTRL)-(D) to log out. If you're using PAM, use (CTRL)-(C). Logging out ensures that no other person can use your account when you leave. (For additional information on making your account secure, refer to the chapter "Keeping Your System Secure" in this manual.)

Using exit

Perhaps the most common way to log out is by using the `exit` command:

```
$ exit
```

Using CTRL-D

Another commonly accepted way to log out is to hold down (CTRL) while pressing (D) ((CTRL)-(D)). On some systems, however, this may not work, and you'll see this message:

```
Use "exit" to logout.
```

Using CTRL-C (with PAM)

If you're using PAM (the Personal Applications Manager), you can exit by holding down (CTRL) while pressing (C) ((CTRL)-(C)).

The "logout" Message

When you log out, HP-UX typically displays a logout message, followed by a new login prompt. Your screen might look something like this after you log out:

```
$ exit
logout

login:
```

At this point, you or any other user can log in.

If none of the methods described above logs you out, consult your system administrator or the manual *HP-UX System Administration Tasks*.

Chapter Command Summary

Table 1-1. Commands

To Do This ...	Type This ...
Log in	(after `login:`) *your_username* (after `Password:`) *your_password* (after `TERM =`) *your_term_type*
Enter a command	(type the command after the prompt and press Return)
Find out your shell name	`echo $SHELL`
Set or change password	`passwd`
Log out	`exit`

2

Working with Files

Conceptually, a **file** is a named container in which you can store information. Files are the basic means which HP-UX uses to organize information. In using HP-UX commands and applications, you create files containing a variety of data. This chapter discusses various aspects of handling files:

- Creating and listing files.
- Naming files.
- Viewing and printing files.
- Renaming, copying, and removing files.
- Finding out who can use your files.

Creating and Listing Files

Using the `cat` command, you can create a file containing text. To list the names of your files, use the `ls` command.

Creating a File with cat

The examples in this chapter and in remaining chapters assume you've created a text file named `myfile`. To create the file, use the `cat` command as follows:

```
$ cat > myfile
_
```

After you type this command, the cursor sits on a line by itself: you can now type text into the empty file. (Press Return at the end of each line you type.) When you have finished entering text, press Return. Then hold down CTRL and press D. The `cat` command stops and returns you to the command line prompt.

You can use the `cat` command to create your own version of `myfile`. For example, you might create the file as follows:

```
$ cat > myfile
The text I am typing will be stored in "myfile."Return
I press RETURN at the end of each line.Return
When I'm finished, I hold down the CTRL key and press D. Return
CTRL-D

$
```

Note You can also create and edit files using a text editor such as `vi`. To learn how to use this editor, see Chapter 6 in this manual, "Writing and Editing with the vi Screen Editor."

Listing Files Using ls

To verify that `cat` created `myfile`, run the `ls` command, which lists the names of your files. Running the `ls` command with the file name will confirm that that file exists, but won't list other files.

```
$ ls myfile          The ls command lists myfile.
myfile
```

(Viewing the file's contents is discussed in the section, "Viewing and Printing Files.")

Naming Files

When you choose a file name, you need to follow certain rules regarding the length of the name and the types of characters you include. If a file name begins with a dot (.), it is "invisible" and the `ls` command normally will *not* list it. To see invisible file names, run `ls` with the `-a` option.

Choosing a File Name

When you choose a file name, remember these rules:

- Generally, file names can contain up to 14 characters (or bytes, in non-ASCII character sets), which can be any combination of the following:
  - Uppercase or lowercase letters (`A` through `Z`; `a` through `z`)
  - Digits (`0` through `9`)
  - Special characters, such as: `+`, `-`, `_`, `.`

  Based on these rules, the following are valid file names:

```
money           Acct.01.87      CODE.c
lost+found      112.3-data      foo_bar
```

- HP-UX interprets uppercase and lowercase letters differently in file names. Thus, the following file names all are different:

```
money      Money      MoneY      MONEY
```

Note Some systems may be configured to accept file names longer than 14 characters. However, before you create files with longer names check with your system administrator. If your system is not configured correctly, the longer file names will be truncated to 14 characters, and may cause difficulties.

Invisible File Names

A file name in which the first character is a dot (.) is an **invisible file name**, since the `ls` command does *not* normally display it. Use invisible file names if you don't want or need certain files displayed when you run `ls`.

To illustrate, you have an invisible start-up file that the system runs when you log in. In HP-UX terminology, this file is called a **login script**. It is used to customize your working environment. To learn more about login scripts, see the chapter, "Customizing Login Scripts," in this manual.

Note that the behavior of your system, including the name of your login script, is determined by the type of command interpreter, or **shell**, that your system is using. Common shells used with HP-UX include the Bourne, Korn, C, and Key Shells. The Key Shell is an interactive shell which uses all the same functions as the Korn Shell.

To force `ls` to list invisible file names, including the name of your login script, run it with the `-a` option:

```
$ ls -a                          Use -a to see invisible file names.
.profile    myfile               This is the Bourne Shell, so .profile is
                                 shown.
```

If your shell is ...	Then your start-up file is ...
Bourne Shell	`.profile`
C Shell	`.login`
Korn Shell	`.profile`
Key Shell	`.profile`
PAM Shell	`.environ`

Viewing and Printing Files

Using the `more` command, you can look at the contents of a text file. If your system is appropriately configured, you can print a text file using the `lp` command.

Viewing a File with more

The `more` command displays a text file's contents on the screen. For example, the following `more` command displays the contents of `myfile` (which you created in the section, "Creating and Listing Files"):

```
$ more myfile
The text I am typing will be stored in "myfile."
I press RETURN at the end of each line.
When I'm finished, I hold down the CTRL key and press D.
$
```

If the file contains more lines than are on your screen, `more` pauses when the screen is full. With a longer file, press space to continue looking at additional screens, and press Q when you are finished. Then `more` returns you to the system prompt.

Try running `more` on the system file `/etc/disktab`:

```
$ more /etc/disktab
# @(#) $Revision  64.5$
#
# This file contains the disktab entries for the current
# sectioning scheme.
#
# Note that the section sizes are in terms of DEV_BSIZE which
# is defined to be 1024 bytes.
#                          .
                           .
                           .
--More--(4%)
```

The "--More--(4%)" message at the bottom of the screen means you've viewed 4% of the file thus far, and 96% of the file remains to be viewed. At this point, you can do any of the following:

- Scroll through the file a page at a time by pressing the space bar.
- Scroll through the file a line at a time by pressing Return.
- Quit viewing the file and leave man by pressing q.

Printing a File with lp

If your system is appropriately configured, you can print a text file using the lp (*l*ine *p*rinter) command. Before using the lp command you may need to find out whether your system is set up so that you can use the lp command. If it is not, you may have to use a command other than lp to get printouts. You can find information on managing printers and other peripherals in *System Administration Tasks*.

If lp does work on your system, you may also need to find out the location of the printer, on an extensive system. When you have this information, print myfile by running the lp command:

```
$ lp myfile
```

If the lp command is working properly, it should display a message indicating that it sent your file to the printer. For example:

```
request id is lp-number (1 file)
```

The *number* is an i.d. number assigned to the print job by the lp command. If you don't see this message, or if you get an error message, consult your system administrator. If lp works successfully, you should get a printout with your username displayed on the first page. The time required for a printout depends on the number of tasks being run by the system and the speed of the printer.

To display a report on the printer status, including the order of your print job in the printer que, type:

```
$ lpstat -t
```

To cancel a print job, enter the cancel command, with the i.d. number for your job:

```
$ cancel request_id
```

Renaming, Copying, and Removing Files

To change a file's name, use the `mv` ("*move*") command; to make a copy of a file, use the `cp` ("*copy*") command; to remove a file, use the `rm` ("*remove*") command. The examples in this section assume you have created the file `myfile`, as described in "Creating and Listing Files."

Renaming Files with mv

Using the `mv` command, you can rename the file `myfile` to `foofile` as follows:

```
$ mv myfile foofile
```

To verify that `mv` renamed the file, use the `ls` command:

```
$ ls
foofile
```

To rename `foofile` back to `myfile`, type:

```
$ mv foofile myfile
$ ls                    Using ls, verify that it worked.
myfile                  It worked!
```

Caution When renaming files, take care not to rename a file to the name of a file which already exists in that directory. If you do this, the file that already has the name will be lost.

For example:

```
$ ls
afile    bfile          If you had these files ...
$ mv afile bfile        And you rename afile to bfile ...
$ ls                    Look what happens ...
bfile                   The previous bfile is replaced with the old afile.
```

(The `mv` command can also be used to move files to different locations on the system. This concept is discussed further in Chapter 3 in this manual, "Using Directories to Organize Your Files.")

Copying Files with cp

Copy a file when you want to make a new version of it while still keeping the old version around. For example, to make a new copy of `myfile` named `myfile2`, type:

```
$ cp myfile myfile2
```

Now when you use the `ls` command, you will see the following:

```
$ ls
myfile      myfile2
```

Use `more` to view `myfile2`. You will find that it is the same as `myfile`.

Caution If you copy a file to an existing file, the existing file will be lost.

Removing (Deleting) Files with rm

If you have files that are no longer needed, you should remove (delete) them. Deleting unnecessary files leaves more room for other files on your system. For example, suppose you've finished using `myfile2`, and it is no longer needed. To remove `myfile2`, type:

```
$ rm myfile2
```

To see that `myfile2` was removed, use `ls`:

```
$ ls      The directory listing shows only the other file now remains.
myfile
```

Permission to Access Files

Files are assigned **access permissions** that control who has permission to read or alter files. If you don't have the necessary access permissions to a file, you may not be able to rename or copy it. If this is the case, the system will display a message indicating that you can't perform the command. You can find out more on access permissions in the next section and in Chapter 12 in this manual, "Making Your System Secure."

Finding Out Who Can Use Your Files

Three classes of users (in various combinations) can access files: *owner*, *group*, and *other*. Each class may access files in various ways: read permission, write permission, and execute permission. Use the `ll` command to view file access permissions.

Access to files is restricted by classes of users. The three basic classes of users are:

- *owner*—Usually the person who created the file (for example, you).
- *group*—Several users who have been grouped together (along with you as the owner of the file) by the system administrator (for example, the members of your department).
- *other*—Any other user on the system.

Each of the above classes can access files in any of these three ways:

- *read permission*—Users with this type of permission can view the contents of a file.
- *write permission*—Users with this type of permission can change the contents of a file.
- *execute permission*—Users with this type of permission can execute (run) the file as a program by typing the file name at the command line prompt.

Using the ll Command to Display File Permissions

The `ll` (*l*ong *l*isting) command displays the permissions for *owner*, *group*, and *other*; `ll` also displays the name of the file's owner and group.

Here is a closer view with all permissions indicated (note that the permissions are in sets of three):

```
rwx   rwx   rwx
 |     |     |
owner group other
```

To see the permissions, owner name, and group name on `myfile`, for example, type the following:

```
$ ll myfile
```

When you press (Return), you should see something like this:

```
-rw-r--r--   1  leslie  users   154        Nov 4 10:18        myfile
permissions     owner    group   size            date          file name
```

The first dash on the left indicates that `myfile` is a file (if `myfile` were a directory, you would see a `d` in place of the dash). The next nine positions indicate read, write, and execute permissions for *owner*, *group*, and *other*. When filled, with all permissions granted for a directory, the permissions look like:

```
drwxrwxrwx
```

If a permission is not allowed, a dash appears in place of the letter.

In the example (`-rw-r--r--`), *owner* (`leslie`) has read and write permission (`rw-`); *group* (`users`) and *other* have only read permission (`r--`).

For More Information ...

To learn more about the `ll` command, refer to the *HP-UX Reference* (the `ll` command is listed under *ls* in section 1 of the *Reference*).

For information on how to change the access permissions on files and directories, see Chapter 12 of this manual, "Making Your System Secure."

Chapter Command Summary

Table 2-1. Commands

To Do This ...	Type This ...
Create a file	`cat >` *filename*
Terminate keyboard input for `cat`	CTRL-D
List visible files in current directory	`ls`
List visible and invisible files in current directory	`ls -a`
View a file	`more` *filename*
Print a file	`lp` *myfile*
Rename ("move") a file	`mv` *fromfile tofile*
Copy a file	`cp` *fromfile tofile*
Delete (remove) a file	`rm` *filename*
Find out access permissions	`ll` *filename*

3

Organizing Files in Directories

After you've used your system for a while, you may start accumulating a large collection of files. With the help of directories, you can organize your files into manageable, logically related groups. For example, if you have several files for several different projects, you can create a directory for each project and store all the files for each project in the appropriate directory.

This chapter discusses the following topics:

- What is a directory hierarchy?
- Determining your location in the HP-UX directory structure.
- Specifying files and directories: absolute path names.
- Specifying files and directories: relative path names.
- Creating directories.
- Changing your current directory.
- Moving and copying files between directories.
- Removing empty directories with the `rmdir` command.
- Using filename shorthand: wildcard characters.
- Finding out who can use your directories: directory permissions.

What is a Directory Hierarchy?

3

Like files, directories are containers. But instead of text or other data, directories contain files and other directories. In addition, directories are hierarchically organized; that is, a directory has a parent directory "above" and may also have sub-directories "below." Similarly, each sub-directory can contain other files and also can have more sub-directories. Because they are hierarchically organized, directories provide a logical way to organize files.

Directories are organized into a structure which resembles an inverted tree. These directories (shown as ovals) usually contain more directories, which in turn enable the branching "tree" structure of a typical home directory:

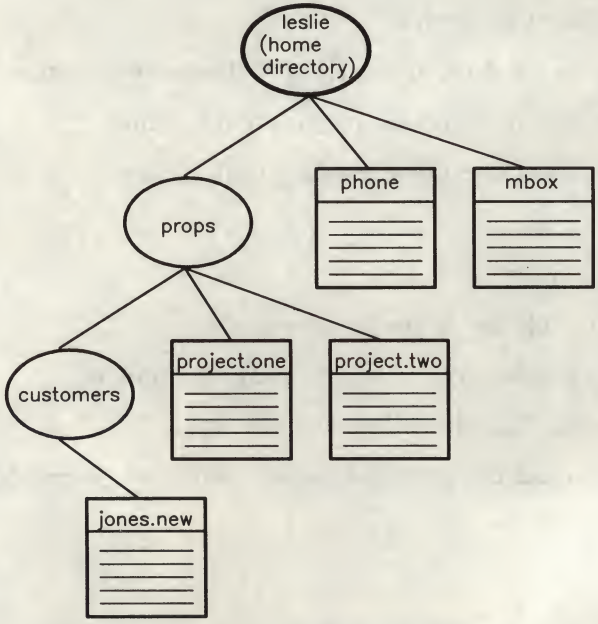


Figure 3-1. A Typical Directory Structure

Each directory also contains files (represented below as boxes), which hold actual text, data, or code. At the top of the inverted tree structure is the **root directory**, represented in path names as /. Figure 3-2 shows a broader part of a system's directory structure.

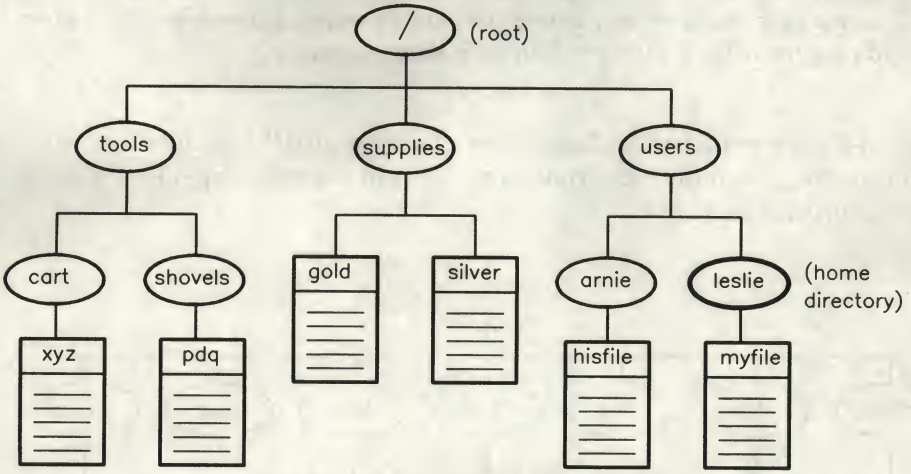


Figure 3-2. A System Directory Structure

Determining Your Location in an HP-UX Directory Hierarchy

This section discusses the HP-UX directory structure and how you specify the location of a file in the structure. All directories fall under the topmost **root** directory, which is denoted by a slash (/). When you use HP-UX, you are working in a directory called the **current working directory**. And when you log in, HP-UX places you in your **home directory**.

Figure 3-3 shows the two highest levels of a typical HP-UX directory structure. Each directory, including the root, may contain logically-organized files, as well as more directories.

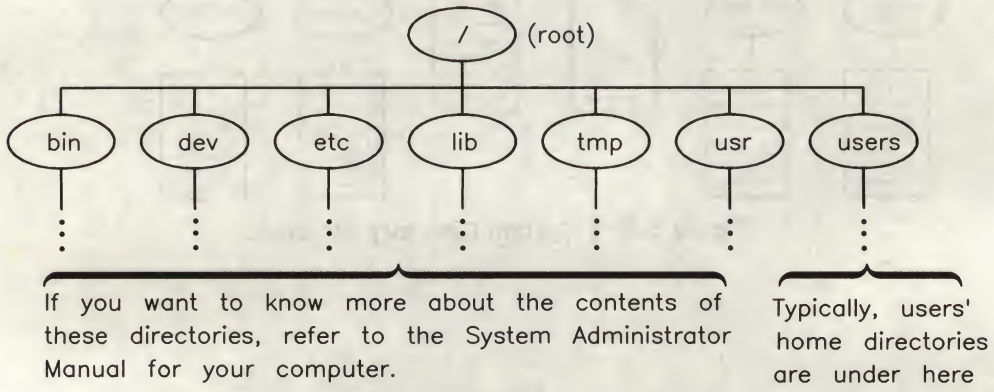


Figure 3-3. The HP-UX Directory Structure

When using HP-UX, you are always positioned "in" a directory. The directory you are performing tasks in is known as your **current working directory**. Moreover, whenever you log in, HP-UX places you in a working directory called your **home directory**.

Here is a sample directory hierarchy for a user named Leslie. When Leslie logs in, she is in her home directory, `leslie`.

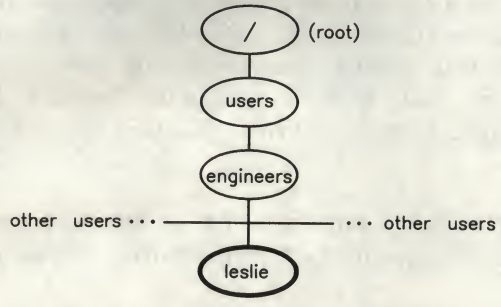


Figure 3-4. Leslie's Home Directory

Specifying Files and Directories: Absolute Path Names

When specifying only files which are in your current working directory, you can refer to them just by their file names. But when referring to directories and files *outside* your current working directory, you must use **path names**, which tell HP-UX how to get to the appropriate directory. An **absolute path name** specifies a path from the root to the directory. A **relative path name** specifies a path from your current directory to another directory.

A path name specifies where a particular file or directory can be found within the directory structure by specifying the directories you need to pass through to get there.

There are two kinds of path names: *absolute* and *relative*.

Absolute Path Names

Absolute path names specify the path to a directory or file, starting from the root directory at the top of the inverted tree structure. The root directory is represented by a slash (/). The path consists of a sequential list of directories, separated by slashes, leading to the directory or file you want to specify. The last name in the path is the directory or file you are pointing to.

To determine the absolute path to your current directory, use the `pwd` (*p*rint *w*orking *d*irectory) command. The `pwd` command displays the "path" from the root directory to your current working directory.

Here is an example of an absolute path, displayed with the `pwd` command:

```
$ pwd
/users/engineers/leslie
```

This specifies the location of the current directory, `leslie`, by starting from the root and working down.

Figure 3-5 shows the absolute path names for various directories and files in a typical directory structure:

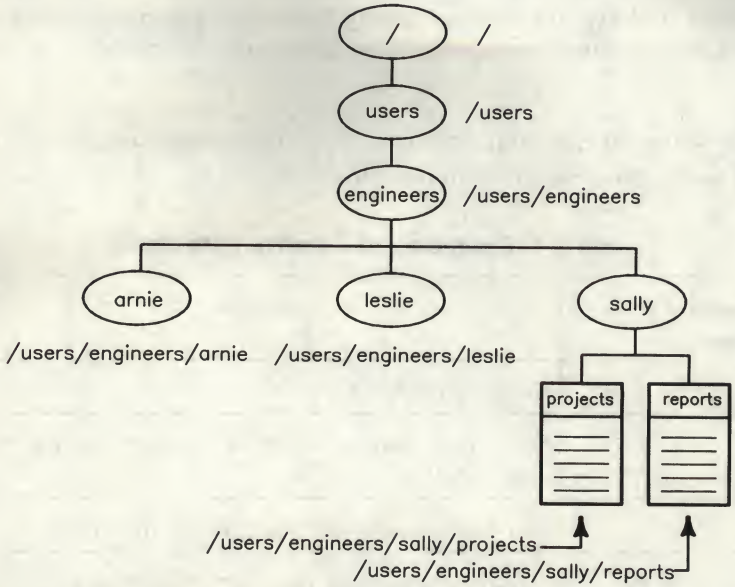


Figure 3-5. Absolute Path Names

Specifying Files and Directories: Relative Path Names

You can use a *relative* path name as a shortcut to the location of files and directories. Relative path names specify directories and files starting from your current working directory (instead of the root directory).

You will frequently find it convenient to use relative path names. The following table shows some common path name shortcuts.

Table 3-1. Examples of Relative Path Names

This relative path name ...	Means ...
.	the current directory.
..	the parent directory (the directory above the current directory).
../..	two directories above the current directory.
directory_name	the directory below the current directory.

For example, suppose the current directory (as shown in Figure 3-6) is `/user/engineers/leslie`. To list the files in the directory above (which is `/user/engineers`), enter:

```
$ ls ..
arnie  leslie  sally   you get a listing of /user/engineers
```

On the other hand, to get a listing of the files in a directory immediately below your current directory, simply enter the directory name. For example, to get a listing of the files in the `projects` directory, below the current directory `/user/engineers/leslie`, you would enter:

```
$ ls projects
$                    The projects directory is empty!
```

Figure 3-6 shows relative path names for various directories and files starting from the current directory, `/users/engineers/leslie`.

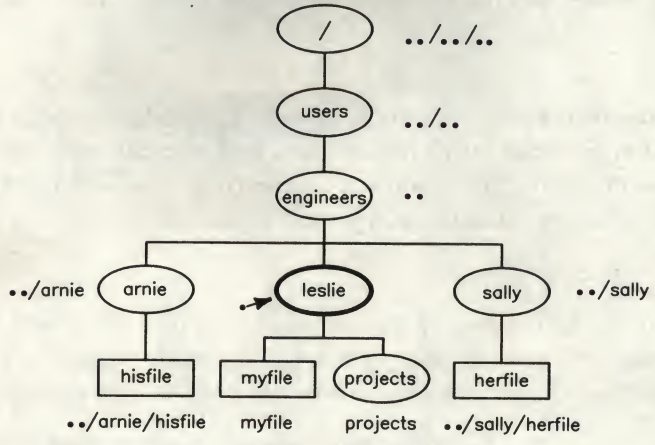


Figure 3-6.
Relative Path Names from
/users/engineers/leslie

Creating Directories

To create a directory, use the `mkdir` command. To get a directory listing that differentiates files from directories, use the `lsf` command instead of `ls`.

The `mkdir` (make directory) command creates a new directory. After you create a directory, you can move files into it, and you can even create more directories underneath it. For example, to create a sub-directory in your current working directory named `projects`, type:

```
$ mkdir projects
```

To verify that it worked, you can use either the `ls` or `lsf` command. Both commands display the new directory, but `lsf` appends a slash (/) to the end of directory names to differentiate them from file names. For example:

```
$ ls                    List files, directories in your current working directory.
myfile  projects        It worked!
$ lsf
myfile  projects/       The lsf command appends a slash to directory names.
```

Figure 3-7 shows the resulting directory structure.

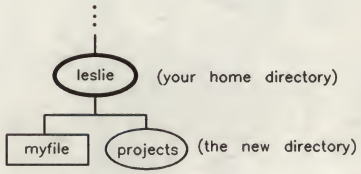


Figure 3-7. Creating the "projects" Directory

The general form of the `mkdir` command is as follows:

`mkdir` *new_dir_path*

where *new_dir_path* is the path name of the directory you want to create. For example, to create a new directory named `old` under the projects directory, type:

```
$ mkdir projects/old
```

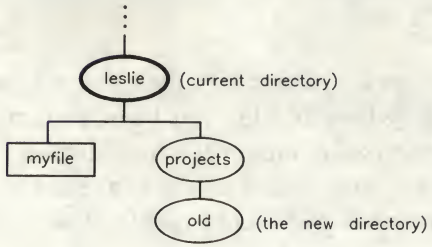



Figure 3-8.
Structure after Creating "old" under the "projects" Directory

Finally, let's create one more directory, named `new`, and verify with `lsf`:

```
$ mkdir projects/new
$ lsf projects
new/      old/          Files and directories are listed alphabetically.
```

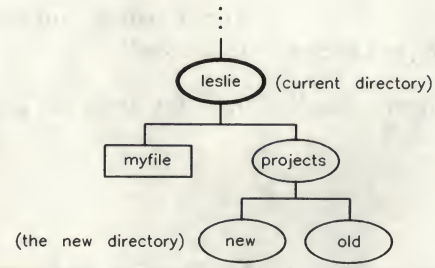


Figure 3-9. Structure after Creating "new" under "projects"

Changing Your Current Directory

Now that you've learned how to create directories under your home directory, you're ready to learn how to move into different directories, using the `cd` command.

3

Using the `cd` ("change directory") command, you can change your current working directory. For example,

```
$ cd projects
```

moves you into the directory `projects` (which you created in the section "Creating Directories"). To verify that you have, in fact, changed your current working directory, use the `pwd` command, which displays your current directory. For example, if your home directory was `/users/leslie`, then, after you run the "`cd projects`" command, `pwd` would display the following:

```
$ pwd
/users/leslie/projects
```

When you're in the new directory, you can list its contents using `lsf`:

```
$ lsf                          Here are the directories you created earlier.
new/      old/
```

To move into the directory `new` under `projects`, type:

```
$ cd new
$ pwd                          Verify where you are.
/users/leslie/projects/new     It worked!
```

Now if you run `lsf`, it won't display anything because there are no files or directories under `new`:

```
$ lsf
$
```

Remember that `..` is the relative path name for the parent directory of your current working directory. So to move up one level, back to `projects`, type:

```
$ cd ..
$ pwd                          Show your current working directory.
/users/leslie/projects         It worked!
```

If you run `cd` without a path name, it returns you to your home directory as the following example illustrates:

```
$ cd
$ pwd              Are you back home?
/users/leslie      Yes!
```


Experiment with the `cd` and `pwd` commands to move around your directory structure. If you become lost, don't panic; just remember that you can run

```
$ cd
```

to return to your home directory. You can also get to any directory using its absolute path name. For example, to change to the `projects` directory in the example hierarchy, enter:

```
cd /users/leslie/projects
```

Figure 3-10 illustrates how various `cd` commands change your current working directory. The example assumes you're starting at the directory `/users/leslie/projects`, and that your home directory is `/users/leslie`.

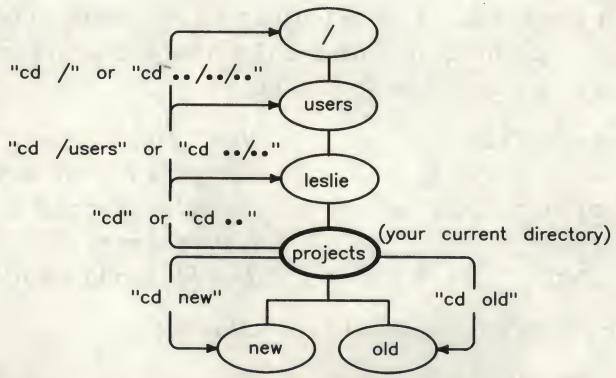


Figure 3-10. Effect of Various "cd" Commands

Moving and Copying Files between Directories

The mv command lets you move a file from one directory to another. With the cp command, you can copy a file into a different directory.

3

Moving Files

In addition to renaming files, the mv command can be used to move files from one directory to another. For example, to move myfile into the projects directory, type:

```
$ cd                                    Move to your home directory first.
$ mv myfile projects
```

Now verify that it worked:

```
$ lsf                                   List your current working directory.
projects/                               Where did myfile go?
$ lsf projects                          Look in the projects directory.
myfile      new/        old/            There's myfile. It worked!
```

Remember that a single dot (.) for a path name represents your current working directory. Therefore, to move myfile from the projects directory back to your current working directory, type:

```
$ mv projects/myfile                    Don't forget the dot.
$ lsf                                   List your current working directory.
myfile      projects/                   It worked; myfile is back.
$ lsf projects                          List projects.
new/        old/                        The file myfile isn't there anymore.
```

The general form of the mv command is as follows:

mv *from_path to_path*

where *from_path* is the file name or path name of the file you want to move, and *to_path* is the name of the path where you are moving the file.

Copying Files

To copy a file into a different directory, use the `cp` command. For example, to make a copy of `myfile` named `myfile2` in the `projects` directory, type:

```
$ cp myfile projects/myfile2
```

```
$ lsf
myfile    projects/     The file myfile still exists.
$ lsf projects
myfile2   new/    old/  The copy (myfile2) is in the projects directory.
```

To make a new version of `myfile2` named `myfile3` in your current directory, type:

```
$ cp projects/myfile2 myfile3
$ lsf
myfile    myfile3    projects/
```

The general form of the `cp` command is as follows:

`cp` *from_path to_path*

where *from_path* is the file name or path name of the file you want to copy, and *to_path* is the path name of the directory or file to which you are copying.

Caution *When moving or copying files, be careful not to destroy an existing file.* For example, if you type the following `cp` command:

```
$ cp myfile3 projects/myfile2
```

Then, a copy of `myfile3` is moved into `projects/myfile2`, overwriting `myfile2`. *The previous contents of* `myfile2` *are lost.*

If you copy a file to a directory, even if the directory has the same name as the file, the directory will not be destroyed. But a file of the same name *in* that directory would be.

As a general rule, before using `mv` or `cp`, use `ls` or `lsf` to ensure that the target file name to which you want to move or copy doesn't already exist.

Removing Empty Directories

When you are finished using a directory and it is no longer needed, you can remove it using the `rmdir` command. For example, if you had a directory containing 5-year-old files on a long-dead project, you might want to remove the files and the directory. Before removing a directory, you must remove its files and subdirectories.

After you've removed a directory, you can no longer use it, and it will no longer appear in an `ll` or other listing.

But before removing a directory, you must remove its files, as well as any directories under it. For example, suppose you want to remove the `projects` directory and the files it contains. Figure 3-11 shows how this structure might look:

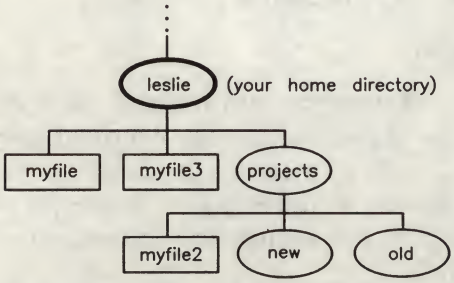


Figure 3-11. The "projects" Directory Structure

To remove this structure, run the following sequence of commands:

```
$ cd                                  Move back to your home directory

$ lsf                                 List the files and directories.
myfile    myfile3    projects/

$ rmdir projects                      Try to remove projects.
rmdir: projects not empty             It won't let you.

$ cd projects                         Change directory to projects.

$ lsf                                 List its contents.
myfile2    new/    old/

$ rm myfile2                          Remove the file myfile2.

$ lsf
new/    old/                          The file myfile2 is gone.

$ rmdir new                           Remove the directory new.  If it's
                                      empty, rmdir removes it.
$ lsf
old/                                  It worked!

$ rmdir old                           Now remove the directory old.  If it's
                                      empty, rmdir removes it.

$ lsf                                 It worked again!

$ cd                                  Now move back to your home direc-
                                      tory ...

$ rmdir projects                      And remove projects.

$ lsf                                 Verify that it worked.
myfile    myfile3                     It worked!
```

File Name Shorthand: Wildcard Characters

Wildcard characters provide a convenient shorthand for specifying multiple file or directory names with one name. Two of the most useful wildcard characters are * and ?. The * matches any sequence (string) of characters (including no characters), and the ? matches any one character.

The * Wildcard

The * wildcard means "any characters, including no characters." Suppose you have created the following files in your current working directory:

```
$ lsf
myfile    myfile2    myfile3    xenic    yourfile
```

To list only the file names beginning with "`myfile`," type:

```
$ lsf myfile*
myfile    myfile2    myfile3
```

Even though `xenic` and `yourfile` exist, `lsf` displays only the file names that start with `myfile`. If you wanted to list file names containing "`file`," type:

```
$ lsf *file*
myfile    myfile2    myfile3    yourfile
```

The ? Wildcard

The ? wildcard means "any single character." Although you probably won't use the ? wildcard as much as *, it is still useful. For instance, if you want to list only the files that start with `myfile` and end with a single additional character, type:

```
$ lsf myfile?
myfile2    myfile3
```

The ? wildcard character matches *exactly one character*. Thus, `myfile` didn't show up in this listing because it didn't have another character at the end.

Using the * Wildcard Character with mv, cp, and rm

Wildcard characters are often useful when you want to move or copy multiple files from one directory to another. For example, suppose you have two directories immediately below your current directory, named `new` and `old`, and these directories contain the following files:


```
$ lsf new
myfile    myfile2
$ lsf old
myfile3   myfile4
```

To move all the files from the directory `new` into the directory `old`, type:

```
$ mv new/* old
$ lsf new                                  The files are no longer in new.
$ lsf old                                  They are in the directory old.
myfile   myfile2   myfile3   myfile4
```

You can do a similar operation with the `cp` command. For example, to copy all the files from `old` into `new`, type:

```
$ cp old/* new
```

Similarly, you can use wildcard characters with the `rm` command.

Caution Be careful when using wildcards that you don't accidentally remove files you need.

For example, to remove all the files in the directory `new`, type:

```
$ rm new/*
$ lsf new
$                    All the files are gone!
```

For More Information ...

See *regexp*(5) in the *HP-UX Reference* for general features of * and ?. For additional features relating to individual shells: if you use the Korn Shell, refer to *ksh*(1); if you use the C shell, refer to *csh*(1), both in the *HP-UX Reference*.

Displaying Directory Permissions: The ll Command

You can display access permissions for a directory, as for a file, with the `ll` command. To display permissions for a specific directory, use the `ll` command with the `-d` option.

To display permissions showing *owner*, *group*, and *other* for a specific directory, use the `ll` command with the `-d` option. For example to see the permissions on the `projects` directory below the current directory, type the following:

```
$ ll -d projects
```

Follow the `ll` *command with a* `-d` *and the directory name.*

When you press (Return), you should see something like this:

```
drwxr-x---  1  leslie  users      1032 Nov  28 12:38 projects
```

The first character (`d`) in the long listing above indicates that `projects` is a directory. The next nine positions (three sets of three) indicate the presence or absence of read (`r`), write (`w`), and search (`x`) permissions for *owner*, *group*, and *other*. If a permission is not allowed, a dash appears in place of the letter. Here is a closer view with all positions indicated:

```
          d      rwx    rwx    rwx
          |      |      |      |
directory owner group other
```

Then, in the original example above (`drwxr-x---`):

owner (`leslie`) has read, write, and search permission (`rwx`); *group* (`users`) has read and search permission (`r-x`); *other* has no access (`---`) to the `projects` directory.

For More Information ...

See Chapter 12 of this guide "Making Your System Secure" for information on how to change the access permissions on files and directories.

To learn more about the `ll` command, refer to the *HP-UX Reference* (the `ll` command is listed under *ls* in section 1 of the *Reference*).

Chapter Command Summary

Table 3-3. Commands

To Do This ...	Type This ...
List files; show directories with "/"	`lsf`
Change directory	`cd` *directory_path*
Change to home directory	`cd`
Display working directory	`pwd`
Remove an (empty) directory	`rmdir` *directory_name*
Display permissions for a directory	`ll -d` *directory_name*

4

Entering Commands: Options, Arguments, and Help with Syntax

HP-UX has many powerful and useful commands. This brief chapter provides some background information that will help you to use these commands more effectively.

More specifically, this chapter discusses the following topics:

- Understanding command syntax—command options and arguments.
- Entering commands with the Key Shell
- Displaying on-line *HP-UX Reference* entries (man pages), if they are installed on your system.

Understanding Command Syntax

HP-UX provides a wealth of powerful, versatile commands that can do a wide variety of computing tasks. Before delving into these commands, though, you should have a general understanding of command syntax.

Most of the commands you've used thus far have been simple in syntax; that is, they've been either a command without any arguments (`whoami`), or a command whose only argument is a file name (`mkdir projects`). In actuality, commands can be more complex, having additional options, arguments, or both. **Options** change a command's behavior. (For example, in Chapter 3, you used the `-a` option to change the behavior of the `ls` command so you could list invisible file names.) In general, command options are preceded by a dash (`-`). **Arguments** provide additional information needed by the command, for example, which files to run the command on.

Examples Using Options

When used without any options, the `rm` command removes a file without verifying whether you really want to remove it. Suppose, for example, your current working directory contains these files: `myfile`, `myfile1`, `myfile2`, `myfile3`, and `myfile4`. You could remove all these files by typing this command:

```
$ rm my*
$                    All the files are removed, no questions asked.
```

For safety, if you want `rm` to prompt you for verification before removing each member of a set of files, you can use the `-i` option:

```
$ rm -i my*
myfile1: ? (y/n)
```

For each file, the system asks if you really want to remove the file. Type `y` *to remove this file;* `n` *to leave it alone.*

```
myfile1: ? (y/n) y
myfile2: ? (y/n) y
myfile3: ? (y/n) y
myfile4: ? (y/n) n
```

You don't want to remove this file, after all.

```
$ ls
myfile4
```

It worked: `rm` *did not remove* `myfile4`.

If you are using `rm` non-interactively and the file does not have write permission (for example, with 444 permission), then a message "*filename:* `444 mode ? (yes/no)`" will be displayed. Respond with `y` if you want to remove the file.

Entering Commands with the Key Shell

The Key Shell (**keysh**) is an extension of the Korn Shell, using softkey menus and context-sensitive help to help you with command options and syntax. The Key Shell automatically translates softkey commands into HP-UX commands when you press (Return) to enter the command line.

Using the Key Shell Displays

4

The Key Shell gives you softkey displays at the bottom of your screen which provide a "menu" of basic Korn Shell commands, along with their options in sequence. You will first see a status line like the following

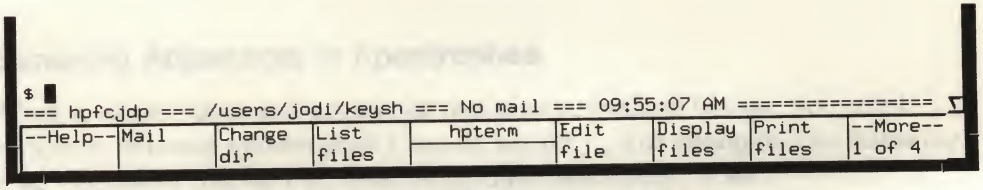


Figure 4-1. Key Shell Softkey Display

You can enter commands from the Key Shell softkey menu or you can enter standard HP-UX commands as usual. If you enter standard HP-UX commands, the Key Shell will often display an appropriate left-to-right set of menu options in the softkey label area at the bottom of your screen. Each label corresponds to a softkey, **f1** through **f8**. The **hpterm** at the center separates the softkeys into groups of four. You may select any or none of the options successively by pressing the corresponding softkey.

When you want to see more commands, or more options to go with a command you've already chosen, press the **--More--** softkey, **f8**. This will cause the Key Shell to display the next "bank" of softkeys in sequence, eventually cycling back to the first, if you press **f8** repeatedly.

After you make a selection by pressing a softkey, your choice will appear on the command line in "English," just as it appeared in the softkey display, with the correct order and spacing.

Entering a Command with the Key Shell

For example, enter the `ls` command. You will see the following:

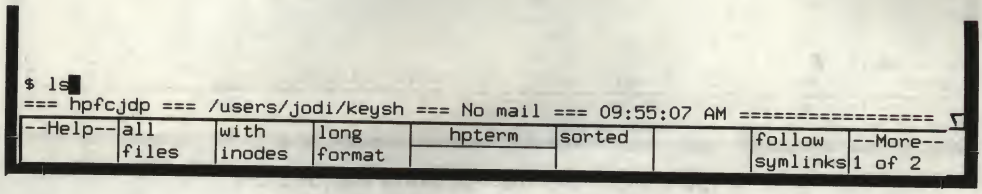


Figure 4-2. Options Displayed

Many softkey commands require that the user enter a parameter or select an additional softkey before pressing (Return). A "prompt line" underneath the command line will indicate whether you need to enter anything else.

If you select `ls`, and then select the "sorted" option, the Key Shell will ask you to specify *how* you want your file listing sorted:

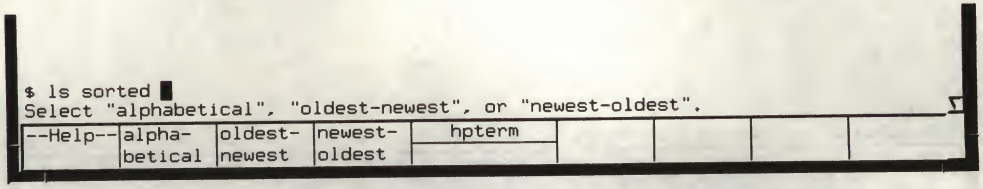


Figure 4-3. Required Options Requested

At any time, you can use the `--Help--` softkey, `f1`, to find out more about what functions are available to you.

Suppose you have selected "newest-oldest" for the `sort` option above. You can now enter the finished command line by pressing (Return). Or, if you want to preview the HP-UX commands to which these "English" words correspond, you can optionally press (Insert line) and the HP-UX commands will be displayed as shown in Figure 4-4.

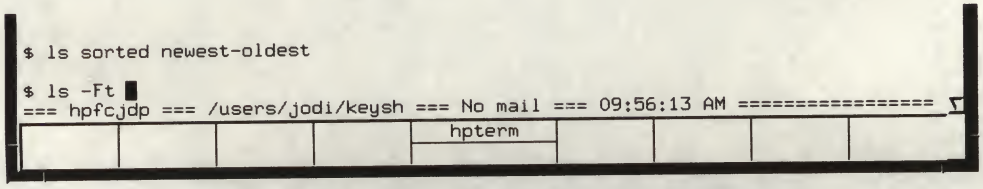


Figure 4-4. Optional HP-UX Commands Display

Customizing Your Key Shell Softkeys

You can change the Key Shell's configuration (for example, status line or options) using the `Keysh_config` softkey in the `--More-- 4 of 4` display. Any changes you make will be automatically saved in the `.keyshrc` file in your home directory. This file will then be replayed upon subsequent invocations of `keysh`.

If they are not already on, some global options that you can change using `Keysh_config` are:

Table 4-1.

To Enable These Options ...	Enter These Softkeys and Press (Return) ...
`--Help--` softkey	`Keysh_config options help on`
Automatic prompt messages	`Keysh_config options prompts on`
Visible HP-UX command translations	`Keysh_config options translations on`

To turn off any of these options, enter `off` at the end of the entry sequence instead of `on`.

Status-Line Indicators you can change, using the `Keysh_config` softkey are:

Table 4-2.

To Enable Additional Status-Line Indicators	Enter These Softkeys and Press (Return)
Host name	`Keysh_config status_line host_name on`
User name	`Keysh_config status_line user_name on`
Current directory	`Keysh_config status_line current_dir on`
Mail status	`Keysh_config status_line mail_status on`
Date	`Keysh_config status_line date on`
Time	`Keysh_config status_line time on`

In addition, the contents of the `$KEYSH` shell variable, if set, is also displayed on the status line.

Restarting and Undoing the Key Shell Configuration Changes

- If you want an existing Key Shell to take on the behavior you've specified in another Key Shell, you will need to enter the following to "replay" the current configuration sequence.

```
Keysh_config restart
```

- You can undo any configuration changes you may have made since last invoking a Key Shell by entering the following command:

```
Keysh_config undo
```

Pressing undo a second time will "toggle" the configuration back to the previous state again, restoring your changes.

Summary of Key Shell Procedures

The general rules for using the Key Shell are:

- Select any desired softkeys from left to right.
- Use the `--More--` softkey to see more options.
- Optionally, use the Insert line key to preview the translated command-line.
- Use the `--Help--` softkey to find out more functions.

If you make an error, use Back space or CTRL-h to erase the line back to where you want to re-enter command text, just as you would with commands in `ksh`, `csh`, or `sh`.

You can also use the arrow keys, Clear line, Delete line, Insert char, and Delete char to manipulate your command line, in addition to using the editor which is set for your Korn Shell (*see* Chapter 9). Note that Clear line functions to delete the line only from the cursor position to the end of the line. Delete line, however, deletes the entire command line and cancels the command.

Accessing On-Line Command Help: man pages

HP-UX commands are documented thoroughly in the *HP-UX Reference*. In addition, HP-UX can also store *HP-UX Reference* entries (commonly known as *man pages*) on disk. If your system has these on-line man pages, you can use the `man` command to display them on the screen.

The `man` (*man*ual) command displays a command's syntax plus a detailed description of the command and its options and arguments (if any). Also, `man` may display examples of command usage and provide other information such as system files used, related commands, diagnostics, possible problems (bugs).

For example, you can use `man` to learn more about the `man` command itself. After a few seconds, an information display appears:

```
$ man man
MAN(1)                                                    MAN(1)

NAME
     man - find manual information by  keywords;
     print out the manual

SYNOPSIS
     man -k keyword ...
     man -f file ...
     man [ - ] [ section ] title ...

DESCRIPTION
     Man is a program that gives information from the
     HP-UX Reference  Manual ...

--More--(11%)
```

The message `--More--(11%)` means you've viewed 11% of the file, and 89% remains. (Some systems will just display `--More--`). At this point, you can do any of the following:

- Step through the file a page at a time by pressing the space bar.
- Scroll through the file a line at a time by pressing Return.
- Quit viewing the reference page by pressing q.

Note that you can only move *forward* through a file.

For details on the syntax for a command, refer to the "SYNOPSIS" section. The above example shows, for instance, that the `man` command actually has three different syntaxes:


```
man -k keyword ...
man -f file...
man [ - ] [section] title ...
```

The brackets, [ ], indicate that the enclosed parameter is optional. For example, when you ran `man man` in the previous example, you used the third syntax, *without* the optional parameters:

```
man [ - ] [section] title ...
```

You should experiment with using the `man` command to learn more about the various commands we've discussed thus far. For instance, to learn more about the `ls` command, type:

```
$ man ls
```

To learn more about the `cp` command, type:

```
$ man cp
```

Chapter Command Summary

Table 4-3. Commands

To Do This ...	Type This ...
Delete (remove) a file interactively	`rm -i` *filename*
Run several commands on same line	*command*; *command2*
Show Additional Key Shell choices	`--More--`
Change Key Shell configuration	`Keysh_config`
Find HP-UX command information on-line	`man` *command_name*

5

Using Commands: Syntax, Input, and Output

HP-UX uses command interpreter programs called shells which help the user interact more effectively with the operating system. This chapter introduces you to some of the useful features of shells.

Specifically, this chapter discusses the following topics:

- Transferring data among files and programs.
- Writing standard output to a file.
- Using files for standard input.
- Redirecting both standard input and standard output.
- Using the output of one command as the input to another: pipes.

Transferring Data Among Files and Programs

The shell interprets your keyboard commands for the HP-UX operating system to act on. When you log in, you are said to be "in" a **shell**. After the shell interprets a command line, HP-UX loads the program that the command has specified into memory and begins running it. When a program is loaded and running, it is called a **process**. HP-UX assigns every process a unique number, known as a **process identifier** (**PID**).

Programs Create Processes

When you log in, HP-UX starts your shell. During login, HP-UX copies the shell program from system disk into memory. When it is in memory, the shell begins executing, and it becomes a process that lasts until you log out. **Process**, then, refers to the copied program that is actively executing in memory, while **program** is the file stored on the disk.

Similarly, the commands you type create processes. After you type a command line, the following events take place:

1. The shell interprets the command line and searches the disk until it finds the requested program.
2. The shell asks HP-UX to run the program; then control transfers from the shell to HP-UX.
3. HP-UX copies the specified program from a disk file into memory. When the program resides in memory, it begins executing—and a process is created.
4. Each process is assigned a **Process Identifier** or **PID**. You can find out what processes are currently running on your system by typing `ps -ef`.
5. When a program finishes executing, control transfers back to the shell, and the process disappears.

Using Standard Input, Standard Output, and Standard Error

Each process opens three standard "files": standard input (`stdin`), standard output (`stdout`), and standard error (`stderr`). Programs use these as follows:

- **Standard input** is the place from which the program expects to read its input. By default, processes read `stdin` from the keyboard.
- **Standard output** is the place the program writes its output. By default, processes write `stdout` to the terminal screen.
- **Standard error** is the place the program writes its error messages. By default, processes write `stderr` to the terminal screen.

Figure 5-1 illustrates the relationship of these files to the process.

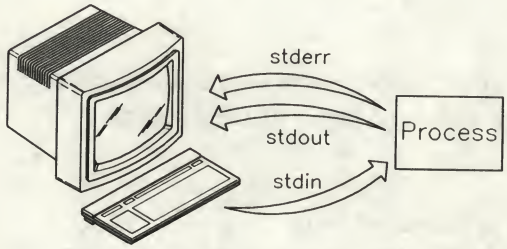


Figure 5-1.
Standard Input, Standard Output, and Standard Error

Examples Using the Standard Files

The following three examples illustrate standard output, standard input, and standard error, respectively:

1. Standard Output

```
$ whoami      The whoami command begins the process.
terry         Standard output is displayed on the screen.
$ _           Control returns to the shell.
```

The above example illustrates standard output. In this example, the `whoami` command uses standard output to display the username of the person typing the command. The prompt returns, indicating that the shell is ready for another command.

5

2. Standard Input

```
$ sort        The sort command uses standard input.
duffy         Enter standard input at the keyboard.
muffy
daffy
CTRL-D        End of standard input.
daffy         Standard output is displayed on the screen.
duffy
muffy
$ _           Control returns to the shell.
```

The above example uses the `sort` command to sort text typed at the keyboard. Typing CTRL-D ends standard input. The standard output is displayed on the terminal screen.

3. Standard Error

```
$ mroe memo          You have misspelled a command.
mroe: not found      Standard error is displayed on the screen.
$ _                  Control returns to the shell.
```

The standard error example illustrates what happens if you misspell a command. The typing error causes the shell to generate an error message that is sent to standard error. Again, the prompt returns, indicating that the shell is ready for another command.

For More Information ...

Your shell has many productivity-enhancing functions that you may find useful after you've mastered the basics. To learn more about the shell processes, including the standard files, see Chapters 9 and 10, and refer to *Shells: User's Guide*.

Writing Standard Output to a File

The shell lets you *redirect* the standard output of a process from the screen (the default) to a file. Redirecting output lets you store the text generated by a command into a file; it's also a convenient way to select which files or devices (such as printers) a program uses. To *redirect* a process's output, separate the command and the output file name with a greater-than sign (>) pointing from the command to the output file. To *append* the output to an existing file, use two greater-than signs (>>) pointing to the file to be appended on.

Redirecting Standard Output

5

You can use output redirection with any command that writes its output to `stdout` (your screen). You can redirect output either to a new file or to an existing file. To redirect output, use a greater-than sign (>). The word following the sign identifies the file name where the `stdout` data is to be written. If the file exists, its previous contents are lost. If the file does not exist, it is created. In its simplest form, the command syntax is as follows:

command > *outfile*

where *command* is the command whose output is redirected, and *outfile* is the name of the file to which the process writes its standard output.

The example below shows output redirection using the `who` command, which displays a list of users currently logged in to the system. Instead of displaying the users on the terminal screen, the output is redirected to the file `whoison`. The `more` command lists the contents of the `whoison` file, showing that the output redirection was successful:

```
$ who > whoison                                    Redirect output to whoison.
$ more whoison                                     Display contents of whoison.
pat          console      Oct  9 08:50
terry        tty01        Oct  9 11:57
kim          tty02        Oct  9 08:13
$
```

Figure 5-2 illustrates where `stdin`, `stdout`, and `stderr` go when output is redirected to a file.

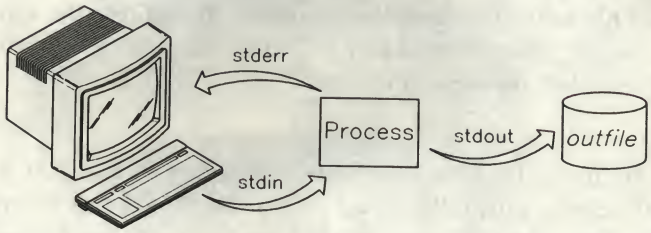


Figure 5-2.
Standard Input, Output, and Error When Output Is Redirected

Appending Output to a File

Output redirection that appends to an existing file uses two greater-than signs (>>). The word following the redirection sign identifies the file to which `stdout` data is appended. If the file exists, the new data is appended to the end of the file. If the file does not exist, it is created. The command syntax is:

command >> *outfile*

where *command* is the command whose output is redirected, and *outfile* is the name of the file to which the process appends the standard output.

The next example executes the `date` command with the output redirected to append to the `whoison` file:

```
$ date >> whoison                          Append output to whoison.
$ more whoison                             Display contents of whoison.
pat         console      Oct  9 08:50      Output from previous example.
terry       tty01        Oct  9 11:57
kim         tty02        Oct  9 08:13
Tue Oct  9 13:20:16 MDT 1990               Newly appended output from
                                           date.
$
```

Using Files for Standard Input

The shell lets you redirect the standard input of a process so that input is read from a file instead of from the keyboard. To redirect the input of a process, separate the command and the input file name with a less-than sign (<) directed at the command name.

Your shell can redirect the data flow into a program so that the keyboard is not used. Input that is normally typed at the keyboard can be redirected to be read from a file. You can use input redirection with any command that accepts input from `stdin` (your keyboard). You cannot apply input redirection to commands such as `who` that do not accept input from `stdin`.

To redirect input, use a less-than sign (<). The word following the sign identifies the file from which the `stdin` data is read. The file must exist for the redirection to succeed. In its simplest form, the command syntax is as follows:

5

command < *infile*

where *command* is the command whose input is redirected, and *infile* is the name of the file from which the process reads standard input.

The following example illustrates input redirection. First, standard output from the `who` command is redirected to a file named `savewho`. Second, the `more` command displays the contents of `savewho`. Finally, standard input for the `wc` (word count) command is redirected to come from the `savewho` file:

```
$ who > savewho                          Redirect output to savewho
$ more savewho                           Display contents of savewho
pat         console      Oct  9 08:50
terry       tty01        Oct  9 11:57
kim         tty02        Oct  9 08:13
kelly       tty04        Oct  9 10:04
$ wc -l < savewho                        Redirect input from savewho
          4
$ _
```

In the preceding example, the `wc` command with the `-l` option counts the number of lines in the input file. Because input is redirected from `savewho`, this number equals the number of users logged in to the system when the `who` command was executed.

Figure 5-3 illustrates where `stdin`, `stdout`, and `stderr` are directed when input is redirected from a file.

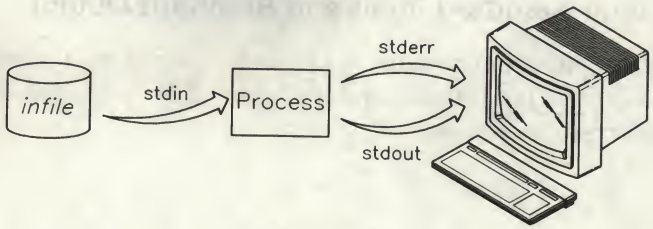


Figure 5-3.
Standard Input, Output, and Error When Input Is Redirected

Redirecting Both Standard Input and Standard Output

You can redirect both the standard input and the standard output of a single command. However, do *not* use the same file name for standard input and standard output. When input and output operations use the same file, the original contents of the input file are lost.

Using the Default Standard Input and Standard Output

The following example uses the `sort` command to sort text typed at the keyboard. Typing CTRL-D ends standard input. The standard output displays on the terminal screen as follows:

```
$ sort
muffy
happy
bumpy
CTRL-D                    End of standard input.
bumpy
happy
muffy                     End of standard output.
$ _
```

Redirecting Standard Input

In the following example, input is redirected:

```
$ more socks              Display contents of socks.
polka dot
argyle
plaid
$ sort < socks            Redirect input from socks and sort the
                          contents.
argyle
plaid
polka dot
$ _
```

In the preceding example, the `sort` command uses a file named `socks` as input. As with the first example, the standard output displays on the terminal screen.

Using Both Standard Input and Standard Output Redirection

The next example combines both input and output redirection:

```
$ sort < socks > sortsocks      Use both input and output redirection.
$ more sortsocks                Display contents of sortsocks.
argyle
plaid
polka dot
$ _
```

In this example, the `sort` command reads input from the `socks` file and writes output to the `sortsocks` file; thus, standard output (unlike the first two examples) does not display on your screen.

Figure 5-4 illustrates where `stdin`, `stdout`, and `stderr` are directed when both output and input are redirected from and to files.

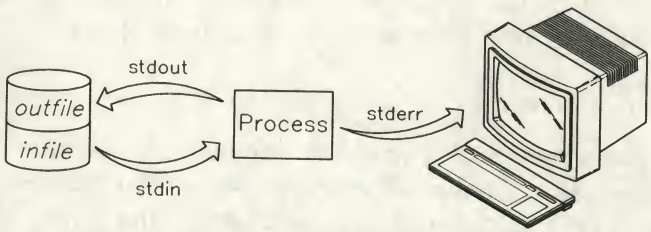


Figure 5-4. Redirecting Both Input and Output

Using the Output of One Command as Input to Another: Pipes

The shell lets you connect two or more processes, so the standard output of one process is used as the standard input to another process. The connection that joins the processes is a **pipe**. To pipe the output of one process into another, you separate the commands with a vertical bar (|).

In previous sections, you were shown how standard output and standard input are redirected to and from files. Using a pipe, output from one command is sent directly to another command as input. A pipe can link any two programs, provided the first program writes its output to `stdout` and the second program reads its input from `stdin`. The general syntax for a pipe is as follows:

5

command1 | *command2*

where *command1* is the command whose standard output is redirected or piped to another command, and *command2* is the command whose standard input reads the previous command's output. You can combine two or more commands into a single pipeline. Each successive command has its output piped as input into the next command on the command line:

command1 | *command2* | ... | *commandN*

You can use pipes whenever you want one command to use the output of another command. In the following example, output from the `who` command is again stored in the file `savewho`. Then, the `savewho` file is used as input to the `wc` command:

```
$ who > savewho       Redirect output of who to file savewho.
$ wc -l < savewho     File savewho is input to wc command.
    4                 Sample result.
```

With a pipeline, these two commands become one:

```
$ who | wc -l
    4
```

As this example illustrates, using pipes eliminates the need for temporary intermediate files. Instead, the standard output from the first command is sent directly to the second command as its standard input.

Using the tee Command with Pipes

The `tee` command lets you divert a copy of the data passing between commands to a file without changing how the pipeline functions. The example below uses the `who` command to determine who is on the system. In the example, which is further illustrated in Figure 5-5, the output from `who` is piped into the `tee` command, which saves a copy of the output in the file `savewho`, and passes the unchanged output to the `wc` command:

```
$ who | tee savewho | wc -l
      4
$ more savewho
pat          console      Oct  9 08:50
terry        tty01        Oct  9 11:57
kim          tty02        Oct  9 08:13
kelly        tty04        Oct  9 10:04
$ _
```

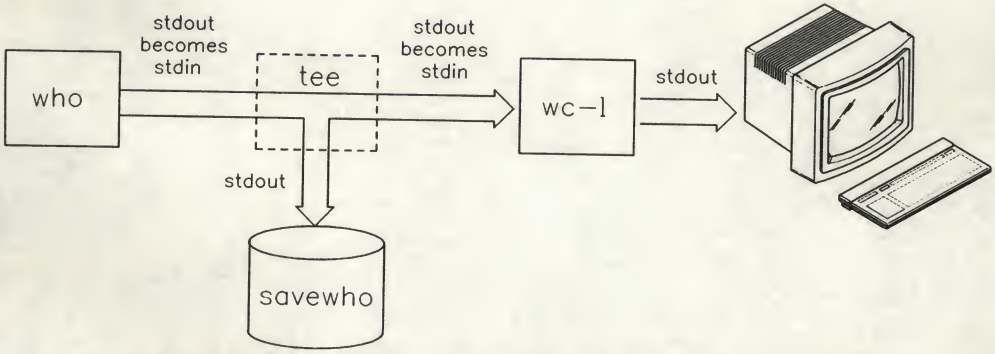



Figure 5-5. Standard Input and Output with Pipes and tee Command

Chapter Command Summary

Table 5-1.

To Do This ...	Type This ...
Redirect (save) standard output to a file	*command* > *outfile*
Append standard output on a file	*command* >> *outfile*
Redirect input from a file to a command	*command* < *infile*
Redirect both standard input and output to a file	*command* < *infile* > *outfile*
Connect ("pipe" between) two process	*command1* | *command2*
Simultaneously file command output and send to another command ("tee")	*command1* | `tee` *file* | *command2*

6

Writing and Editing with the vi Screen Editor

As you look through this chapter, you will learn how to use `vi` to do the following tasks:

- Use the keyboard with `vi`..
- Enter commands versus type text (the `vi` *modes*).
- Do essential tasks with a file.
- Enter and delete text and undo a command.
- Move and copy text
- Save and print a file; exit `vi`.
- Move the cursor.
- Scroll through text.
- Find text patterns.
- Replace and substitute characters.
- Use options to change your `vi` environment.
- Make your `vi` environment permanent.

You will probably want to refer to the `vi` *Editor Quick Reference* which you can tear out of the front of this manual.

Using the Keyboard with vi

The `vi` "Visual Interactive" editor has two basic modes for manipulating text:

- Command mode
- Text entry mode

When you enter `vi`, you will be in command mode until you enter one of the text entry codes, such as `i`, `a`, `s`, or `o`, which are explained in this section. In text entry mode, you can backspace and type over text you have just entered (by pressing CTRL-h or Back Space). But, if you want to move around otherwise in your text and execute other text-manipulation commands, you will have to press ESC to return to command mode.

If You Make Mistakes

Use the following procedures to correct mistakes:

- If you type an error while entering text, press Back space to back up over the error, and then re-type the correct text.
- If you type several errors and cannot recover, exit `vi` without saving the file, and start over. To do this, press ESC. Then type:

  `:q!` Return

- For other situations, see the section "Undoing Mistakes," later in this chapter.

A Note about Letter Case

When you enter commands in `vi`, letter case (caps or small letters) *does* matter. For example, lowercase `i` and uppercase `I` represent two different commands. Therefore, if the cursor doesn't move as it should, make sure the Caps key isn't locked on, or see your system administrator.

Overview: Entering Commands vs. Entering Text

> The vi editor lets you either type text or type commands. Because vi doesn't let you do both at the same time, you will find yourself alternating between two modes. You type text in *text-input mode*. You type commands, move the cursor, set options, and exit vi in *command mode*. To return to command mode from text-input mode, press ESC.

To enter vi, type vi followed by a space and the name of the file you want to work with (either a new one or one that already exists). You automatically enter vi in *command mode*—which means you can type commands, but you can't type text.

For example, you can type commands to move the cursor or to exit vi. To begin typing text, press a key such as i that lets you enter text. When you finish typing text and want to type commands again, press the ESC key.

Distinguishing between Modes

6

If you aren't sure which mode you are in, press ESC:

- If you are in text-input mode, pressing ESC puts you into command mode.
- If you are already in command mode, pressing ESC should make your terminal beep, if it is so equipped.

The :set showmode option offers an alternative way to distinguish between modes. Setting this option usually displays an INPUT MODE message at the bottom of your screen each time you enter text-input mode.

If you use the showmode option, set it each time you enter vi. To set the showmode option, press ESC to ensure you're in command mode, and then type:

```
:set showmode Return
```

Typing a colon (:) while in command mode opens a command line at the bottom of your screen. The text you type after the colon is a command. Execute the command by pressing Return.

To undo the showmode option, type:

```
:set noshowmode Return
```

Text Input Mode: Entering Paragraphs

When you are in text-input mode, `vi` requires you to end each line by pressing Return. If you don't press Return, the line *wraps* (continues) to the next line—without neatly breaking words. Not entering carriage returns eventually results in the following error message:

```
Line too long
```

If you receive the above error message, type:

```
vi Return
```

After you type `vi` Return, *the line on which you were working will be lost.*

If you don't want to insert carriage returns, use the `:set wrapmargin` option. This option inserts carriage returns automatically. The wrapmargin option is most useful when typing paragraphs, not when typing computer programs.

To *remove* carriage returns, position the cursor on the line having the return character at the end and press J. The J command "joins" lines of text.

Set the wrapmargin option each time you enter `vi`. To set the wrapmargin option, press ESC to ensure that you're in command mode, and then type:

```
:set wrapmargin=n Return
```

The *n* represents a number that you specify and determines the size of the right margin (the default is zero). For example, to set a 72-character line on an 80-character display, type:

```
:set wrapmargin=8
```

To undo the `wrapmargin` option, type:

```
:set wrapmargin=0
```

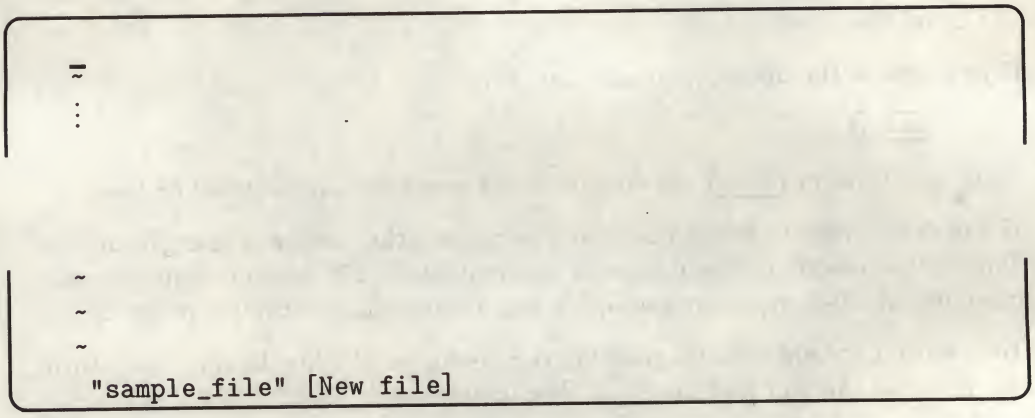
For a more detailed discussion of the `showmode` and `wrapmargin` options, and how to avoid having to set options each time you enter `vi`, please refer to the later sections in this chapter, "Using Options to Change Your vi Environment" and "Making Your `vi` Environment Permanent."

Doing Essential Tasks with the vi Full-Screen Editor

1. First, log in. When you see the HP-UX prompt (usually $ or %), type:

   ```
   $ vi sample_file Return
   ```

   You should soon see the following screen, with the cursor at the top left (`vi` is in command mode when you enter):



   If your screen does not look like the above example, you may have entered an existing file. Type `:q!` if you wish to leave `vi` and return to the HP-UX prompt.

2. Press i ("insert before the cursor") to start entering text in `vi`. You are now in *text-input mode*. The screen will not register any command-mode input, so press the key only once. (Do not press Return.)

3. Type the following text as it appears. Note the mistakes and include them; you will correct them later. If you type a mistake of your own, use the Back space key to move back over the incorrect text, and then retype it.

```
Type this sample text. Return
The second lin contains one error. Return
The third line is fine. Return
But the fourth line cotains two misstakes.
```

4. Press ESC (the escape key). Pressing ESC means you can no longer enter text. Instead, you can *command* the cursor to move within the file (that is, you are in *command mode*). If you aren't sure whether you are in text-entry mode or command mode, press ESC. If you were already in command mode, pressing ESC should make your terminal beep (if your terminal does not have audio, pressing ESC still puts you in command mode).

5. Type :wq Return Typing a colon (*:*) while in command mode opens a command line at the bottom of your screen. The colon and the rest of your command (in this instance *wq*) appears in this command line. When you type the `:wq` command, the *:w* writes *sample_file* to a disk (saves the changes); the *q* tells `vi` to quit.

6. Re-enter *samplefile* to correct the mistakes by typing this command:

```
$ vi sample_file Return
```

This command is the same as before, but instead of opening a new text file, vi finds the existing file called *sample_file*. The screen clears, and you see something like this:

```
Type this sample text.
The second lin contains one error.
The third line is fine.
But the fourth line cotains two misstakes.
~
:
~
~
~
"sample_file" 4 lines, 125 characters
```

6

- To correct the two mistakes (*cotains* and *misstakes*) in the fourth line, do the following:

  1. Press (j) three times to move the cursor *down* to the first character in the fourth line.
  2. Press (l) (lowercase *L*) to move the cursor *right*. Press (l) several times until the cursor is at the *t* in *cotains*. (On most keyboards, holding down a cursor movement key moves the cursor until you release the key.) If you pass *t*, press (h) to move the cursor *left*.
  3. Press (i) to insert text *before* the cursor.
  4. Type in the n to spell *contains* correctly.
  5. Press (ESC). Pressing (ESC) puts you back in command mode, letting you move the cursor without entering text.

- Next, correct *misstakes*, which also is in the fourth line:

  1. Press [l] (lowercase *L*) until the cursor is at the first *s* in *misstakes*.
  2. Press [x] (lowercase *x*) to *delete* the extra *s*. If you make mistakes, such as deleting more than one character, press [U] (uppercase *U*) to undo all the changes you made to that line. After deleting text, you don't need to press [ESC] because you haven't left command mode.

- Now, correct the mistake (*lin*) in the second line:

  1. Press [k] twice to move the cursor *up* to the second line. The cursor should be at the end of the second line.
  2. Press [h] to move the cursor *left*. Position the cursor at the *n* in *lin*. If you pass the *n*, press [l] (lowercase *L*) to move the cursor *right*.
  3. Press [a] to **append** text *after* the cursor.
  4. Type `e` to spell *line* correctly. (Whatever you type is added to the text, until you press [ESC] again.)
  5. Press [ESC] to exit text-entry mode and re-enter command mode.

- You have now corrected the mistakes in *sample_file* and are ready to exit `vi`. You can exit `vi` in one of two ways:

  1. To exit `vi` saving your corrections, type :wq Return.
  2. To exit `vi` *without* saving your corrections, type :q! Return.

Summary of vi Essentials

You should now know how to enter `vi`, enter text, move the cursor, make corrections, and exit `vi`—with or without saving the text. The table below summarizes the basic commands.

Table 6-1. vi Essentials Summary

To Do This ...	Type This Command ...
Enter `vi` and create, or use existing, *sample_file*.	vi *sample_file* Return
Insert text before the cursor.	`i`
Exit text-entry mode and return to command mode.	ESC
Write the file and then quit `vi`.	`:wq` Return
Move the cursor right.	`l` or ▶
Move the cursor left.	`h` or ◀
Move the cursor up.	`k` or ▲
Move the cursor down.	`j` or ▼
Append text after the cursor.	`a`
Delete one character.	`x`
Exit `vi` without saving changes.	`:q!` Return

You have now gone through most of the essentials for doing basic tasks in `vi`. The next sections will give you more details about using this highly versatile editor, beginning with the section "Entering, Deleting, and Undoing".

Entering, Deleting, and Undoing

The vi editor offers various ways to enter text. You can *insert* text before the cursor, *append* text after the cursor, or *open* new lines for inserting text either above or below the cursor. When you finish typing text, press ESC to end text entry mode and return to command mode. See Table 6-2 and Figure 6-3 for summaries of the text-entry commands described in this section.

The commands that let you enter text are the following:

Table 6-2.

To Do This ...	Type This ...
Insert text before the character that cursor is on.	i
Insert text before the first character of the line.	I
Append text after cursor.	a
Append text at the end of the line.	A
Open a blank line below cursor for text entry (lower-case o).	o
Open a blank line above cursor for text entry (upper-case O).	O

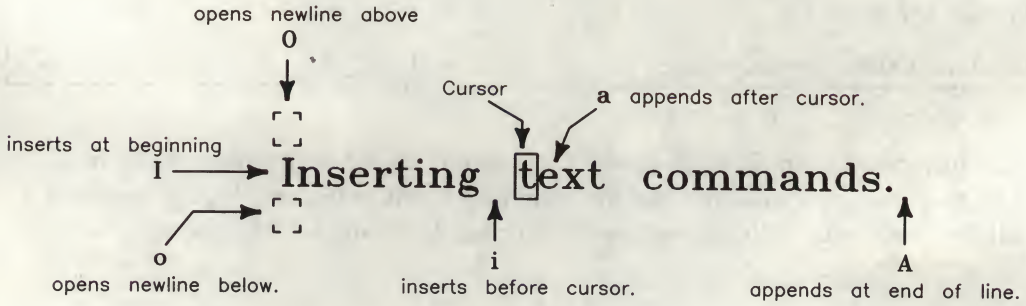


Figure 6-1. Using the Text-Entry Commands

Deleting Text

In `vi`, you can *delete* text by character, word, or line. You can also delete text from the cursor back to the first line of text and from the cursor forward to the last line of text. Use delete commands only in command mode.

The following table summarizes the delete commands.

Table 6-3.

To Delete ...	Type This ...
Character at cursor.	`x`
n characters, starting at the cursor.	*n*`x`
From cursor to beginning of next word or first punctuation.	`dw`
From cursor to beginning of next word.	`dW`
From cursor to beginning of *n*th word or punctuation.	*n*`dw`
Current line.	`dd`
The next *n* lines, including current line.	*n*`dd`
From cursor to end of current line (deletes character at cursor).	`d$`
From cursor to beginning of current line (leaves character at cursor).	`d0` (zero)
All lines to beginning of file, including current line.	`d1G`
All lines to end of file, including current line.	`dG`

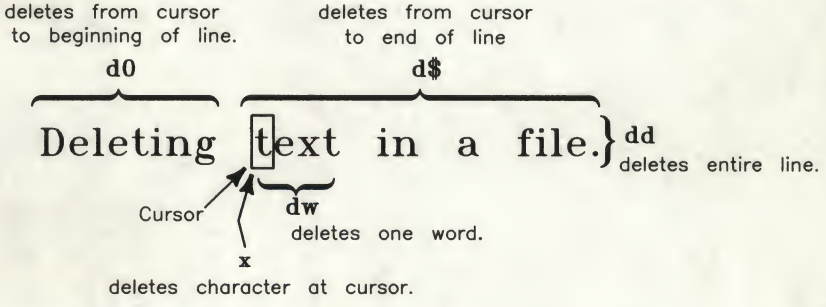


Figure 6-2. Using the Delete Commands

Undoing Mistakes

The `u` undo command (lower-case *u*) reverses the *last* change made to your text. The `U` undo command (upper-case *U*) reverses all the changes made to a *single line* since you began editing that line. Typing `:q!` exits `vi` without saving the text. Use these commands only in command mode.

The following table summarizes the commands you can use to recover from mistakes:

Table 6-4.

To Do This ...	Type This ...
Undo either the *last* command executed or the last characters typed.	`u`
Restore the *current line* to its state before the last edits.	`U`
Exit `vi` without saving changes.	`:q!`

Moving and Copying Text

One of the more useful tasks you can do with the `vi` text editor is to take a piece of text to another place in the file, or copy a piece of text to one or more other locations.

To move a block of text from one location to another, use the *delete* and *put* commands. To copy a block of text from one location to another, use the *yank* and *put* commands.

To move text from one location to another, follow these steps:

1. Press (ESC) to enter command mode.
2. Delete, with a single delete command, all the text you want to move. A line-delete command can be used with multiple lines by preceding it with the number of lines you want to delete, *n*`dd`. Other delete commands can also be used. After deleting the text, don't execute any commands but cursor-movement commands until you're ready to restore, or "put" the text in another location.

3. Move the cursor to the location where you want to put the text you just deleted.
4. Execute a *put* command `p`, which places the deleted text below the cursor, if you have used line delete, `dd`.

An Example of Deleting and Putting Text

For example, suppose you have the following text, where Line Two is in Line Four's place and *This* is misspelled:

```
This is Line One.
This is Line Three.
This is Line Four.
hTis is Line Two.
```

To move Line Two to its place between Lines One and Three, execute the following steps (if you make a mistake, type u to undo the mistake.

1. Place the cursor anywhere on Line Two (*hTis is Line Two*), and type dd. This command deletes the entire line.

   The sample lines now look like this:

   ```
   This is Line One.
   This is Line Three.
   This is Line Four.
   ```

2. Move the cursor up, and position it anywhere on Line One.

3. Type p (lower-case *p*) to put the deleted text *after* Line One. The cursor should be at the first character (*h*) of Line Two (*hTis is Line Two*). The text now looks like this:

   ```
   This is Line One.
   hTis is Line Two.
   This is Line Three.
   This is Line Four.
   ```

4. To correct the transposition error (*hTis*) in Line Two, place the cursor on the "h" and, also in command mode, type xp. Typing x deletes the *h*; typing p puts the *h* after the cursor.

Copying Text Without Deleting

To copy text, and place it at one or more other locations without deleting the first occurrence, follow these steps:

1. Press (ESC) to enter command mode.
2. Position the cursor on the first line of the text you want to copy.
3. Type yy (or *n*yy, for multiple lines) to copy (or "yank") the text you want to move.
4. Move the cursor to the location where you want to put the yanked text.
5. Type p (lower-case *p*) to put yanked text below the cursor.
6. If you wish, you can then move the cursor to yet another location and put the same text there. As long as you haven't executed any intervening commands, the text remains in the "put" buffer.

An Example of Yanking and Putting

For example, suppose you have the following four lines of text:


```
This is the First Line of text you want to copy.
This is the Second Line.
And the Third Line.
This is the Final Line of text you want to copy.
```

To duplicate the above lines of text immediately *after* the original lines, execute the following steps:

1. Position the cursor anywhere on the First Line.
2. Type this command to copy the four lines of text:

   ```
   4yy
   ```

3. Move the cursor down to the Final Line.
4. Type p (lower-case *p*) to put the copied lines of text immediately *after* the Final Line.

The lines of text now should look like this:

```
This is the First Line of text you want to copy.
This is the Second Line.
And the Third Line.
This is the Final Line of text you want to copy.
This is the First Line of text you want to copy.
This is the Second Line.
And the Third Line.
This is the Final Line of text you want to copy.
```

Saving Text, Printing, and Exiting vi

While editing, it is good practice to save the text you are working on. You should do this fairly often in order to protect yourself from accidental loss of a significant amount of data. You may want to save it either to your current text file or to a different text file. When finished editing, you can exit, saving changes or not. You will use the : (colon) command line at the bottom of the screen to execute saves and to "escape to a shell" in order to print a file. Table 6-10, at the end of this chapter, summarizes the commands to save text and to exit `vi`.

Save frequently during editing sessions—particularly after each significant change. To save during an editing session, first press ESC to ensure that you're in command mode, and then type:

`:w` Return

Typing a colon (:) while in command mode automatically opens a command line at the bottom of your screen. The text you type after the colon is a command. Press Return to execute the command.

6

Saving to a Different File

To save to a different text file, use the following command:

`:w` *file_name* Return

For example, suppose you are editing a file called *practice*, and want to save the text to a file called *practice.tmp*. Type:

`:w practice.tmp` Return

`vi` saves the text from *practice* (the file you are currently editing) to *practice.tmp*. Your original *practice* file remains as it was the last time you saved to it.

If the text file you specify for *file_name* doesn't exist, `vi` creates the file. If the file already exists, this error message appears at the bottom of your screen:

`File exists - use "w!` *file_name*`" to overwrite`

To *overwrite* (replace) the contents of *file_name* with the file you're currently editing, re-type the command as follows:

:w! *file_name* Return

Saving Part of a File

To save part of your current text file, to *file_name*, use this command:

:*x,y* w *file_name* Return

where *x,y* represents specific line numbers. You can read the line number where the cursor is located by pressing CTRL-G.

For example, to save lines 20 through 74 of your current text file to a file called *temp*, type:

:20,74 w temp Return

To overwrite (replace) the contents of *file_name*, type a "!" after the *w*:

:*x,y* w! *file_name* Return

To append (add) the contents of part of your file to another file, *file_name1*, without overwriting that file, use the line numbers as above and type >> after the w:

:*x,y* w >> *file_name* Return

Printing a File

The `vi` editor does not support printing a file directly, nor will it directly activate other HP-UX commands while you are working within a file. However, instead of leaving and re-entering `vi`, it is possible to "escape to a shell" and to work temporarily in the system environment. One use of the shell escape is in printing a file, either the one you are currently in or another one.

To print a file from within a `vi` file, press (ESC) and : (colon), as though you were going to type a colon command. Then type ! and the appropriate command.

For example, to print your current file, enter the following:

```
:!lp %
```

To print any file, enter:

```
:!lp filename
```

Be sure you include the path with the *filename*, if the file is located outside your current directory.

The `lp` program gives you a message confirming your printing job number. To resume work, press (Return) when you are prompted to do so. Your cursor will be at the same place you left it.

Exiting vi

When you want to exit `vi`, you can either save or not save your changes:

- To exit `vi` saving the text and all your changes, type:

  `:wq` Return

- To exit `vi` *without* saving your changes, type:

  `:q!` Return

Moving the Cursor

When you are in command mode, `vi` provides various ways to move around in a file. For example, the keys h, j, k, and l (lower-case *L*) move the cursor left, down, up, and right, respectively. Most terminals let you use the arrow keys (the keys marked ◀, ▼, ▲, and ▶) to perform the same function. If the cursor doesn't move as you believe it should, make sure the Caps key isn't locked. Table 6-5 summarizes many of the cursor-moving commands; some of these commands are illustrated in Figure 6-3.

Moving the Cursor to a Specific Line

A useful cursor-moving command is `G`, which functions as a "goto" command. To illustrate, suppose you are editing a file and want to go to line 799 of that file. Type `799G`. The cursor moves to line 799 (you don't need to press Return). Similarly, to go to line 1 of the file, type `1G`. (*Note:* To find your current line number, press CTRL-G; to display line numbers along the left margin of your file, type `:set number`.)

To move the cursor to the last line, simply type `G`.

Table 6-5. Moving the Cursor Up, Down, Left, and Right

To Move the Cursor ...	Type This ...
Left.	h or ◀
Down.	j or ▼
Up.	k or ▲
Right.	l or ▶
To end of current line.	$
To beginning of current line.	0 (zero)
To first character in current line.	^ (zero)
To beginning of next word or next punctuation	w
To last line of file.	G
To specific line of text.	*n*G
To the point of previous change.	'' (two single quotes)

To practice all the line-movement commands, try moving the cursor on a sentence like the following:

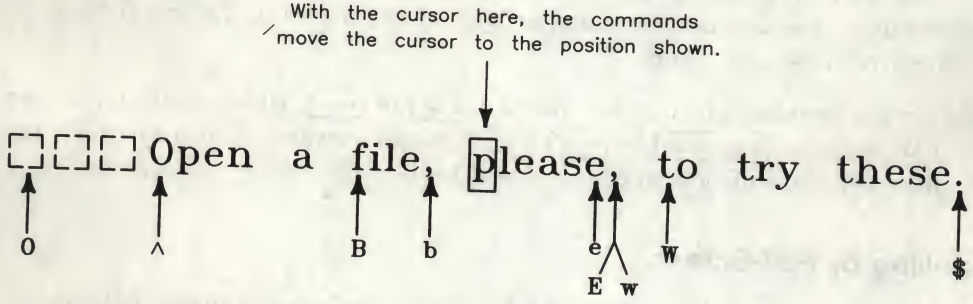


Figure 6-3. Moving the Cursor

Scrolling through Text

In `vi` you can scroll backward or forward by line, half-screen, or full-screen. To scroll, type the appropriate key while holding down the CTRL key. Execute scrolling commands only in command mode. Table 6-6 summarizes the scrolling commands.

Scrolling moves the cursor backward or forward through the text, so you can view various lines or sections of text. Scrolling *backward* lets you view lines toward the first line of text. Scrolling *forward* lets you view lines toward the last line of text.

To execute the scrolling commands, press ESC to ensure that you're in command mode. Then press the CTRL key and, while holding CTRL down, type the appropriate command key—much like holding down the Shift key and typing the key you want to capitalize.

(This guide describes the *default* CTRL key commands. Depending on your terminal type, commands that use the CTRL key may not function as described. If you have problems, see your system administrator or refer to *HP-UX System Administration Tasks*.)

You may want to open a practice file and experiment with the scrolling commands so you can become familiar with them. Refer to Table 6-6 for a summary of these commands.

Note: Some terminals have a key labeled Page or Prev, which pages backward, and a key labeled Page Down or Next, which pages forward. You may prefer to use these keys in addition to conventional keyboard scrolling commands.

Scrolling by Full-Screen

Among the most useful scrolling commands are two which cause the display to scroll backward or forward a full screen, while the cursor takes a constant location on the screen:

- CTRL-B scrolls *backward* one full screen, so you are one screen closer to the beginning of your file. The cursor remains on the bottom line of your screen.

- CTRL-F scrolls *forward* one full screen, so you are one screen closer to the end of your file. The cursor remains on the top line of your screen.

Table 6-6. Using the Scrolling Commands

To Scroll ...	Type This ...
Backward to previous screen.	CTRL-B
Forward to next screen.	CTRL-F

Finding Text Patterns

You can search through your `vi` file for a particular word or words, or a sequence of characters. To search forward from the cursor, type a slash (/) followed by the pattern you want to find. To search backward from the cursor, type a question mark (?) followed by the pattern you want to find. Table 6-7 illustrates some sample search patterns.

To search *forward* from the current cursor position, use this command:

/pattern Return

where *pattern* represents the specific sequence of characters you want to search for.

To search *backward* from the current cursor position, use this command:

?pattern Return

When you press Return, `vi` searches for the pattern you specified. If `vi` finds the pattern, it positions the cursor at the first character in the pattern sequence. For example, to search forward for the word "place," type:

```
/place Return
```

If `vi` finds "place," it positions the cursor at the "p." To search for additional occurrences of "place," press either n or N:

- n continues searching in the same direction for *place*.
- N *reverses the direction* of the pattern search.

If `vi` *does not* find the pattern you specified, it displays the following message at the bottom of your screen, and the cursor isn't moved:

```
Pattern not found
```

Searching for Special Occurrences

In the previous example, `vi` finds *any* sequence containing the pattern "place," including "displace," "placement," and "replaced."

- To find the *single word* "place," type the pattern with a space before and after it (the ␣ represents a space):

  `/␣place␣` Return

- To find *place* occurring *only at the beginning of a line*, precede the pattern with a caret (^):

  `/^place` Return.

- To find *place* occurring *only at the end of a line*, follow the pattern with a dollar sign ($):

  `/place$` Return.

To search literally for such a character as a caret (^) or a dollar sign ($), precede the character with a backward slash (\). The backward slash tells `vi` to search for a special character.

Special characters are those (such as ^ , $, * , / , and .) that have special functions for `vi`. For example, a $ normally means "go to the end of the line," but if the $ is preceded immediately by a \, the $ is simply another ordinary character. Table 6-7 shows examples using the \.

Table 6-7. Example Pattern Searches

Sample Pattern	Explanation
?␣test␣ Return	Searches backward for the word *test*. The spaces (represented in this table by ␣) require that *test* not be part of another word. This search will not find *test* if it is followed by punctuation.
/(No \$ money) Return	Searches forward for the pattern *(No $ money)*. The escape character (\) immediately preceding the $ tells `vi` to search literally for a dollar sign.

Replacing and Substituting Characters

In previous sections, you learned how to move the cursor to an error and correct it. Now you can use a *replace* or *substitute* command to substitute characters and to correct errors. Use these commands only in command mode.

Replacing Single Characters

To *replace* a single character of text, press ESC to enter command mode, position the cursor over the character you want to replace, and type `r` (lower-case *r*) while in command mode. Then type the replacement character. The `r` command lets you substitute *only one* character. After you have replaced the character, you are back in command mode.

The table below illustrates the `r` command:

Original Text	Keystrokes	Modified Text
`This wors needs correcting.`	rd	`This word needs correcting.`

Substituting Characters

To *substitute* one or more characters for a *single character*, type `s` (lower-case *s*) while in command mode. Unlike the `r` command, the `s` command lets you substitute more than one character for a single character.

When you type the `s` command, a dollar sign ($) appears in place of the character. After you type the desired character (or characters), press ESC.

To substitute several original characters with different characters, precede the `s` command by the number of characters you want to substitute.

The table below illustrates the `s` command:

Original Text	Keystrokes	Modified Text
`Here is some sample text.`	sWA (ESC)	`Here WAs some sample text.`
`Edit this lines.`	2sREE (ESC)	`Edit thREE lines.`

The table below summarizes the replace and substitute commands. You may want to open a practice file and experiment with these commands.

Table 6-8. How to Modify Text

To Do This ...	Type This ...
Replace the single character in the cursor position with one new character.	r*character*
Substitute one or more characters for the existing character(s).	s*characters* (ESC)
Substitute characters for the existing *n* character(s).	*n*s*characters* (ESC)

Using Options to Change Your vi Environment

To affect the overall operation of `vi`, you can set (or unset) any of several *options*. When you enter `vi`, options are assigned certain *default* values. To set editor options as you want them, use the `:set` command.

When exiting `vi`, all options return to the default, so you will need to reset your options each time you enter `vi`. See the next section for how to make your options permanent.

To see all your default options, type:

```
:set all
```

By using the `set` command, you can change these defaults. The syntax of the `set` command is as follows:

`:set` *option* Return

where *option* is the name of the editor option you want to use (see Table 6-9 for descriptions of some of these options).

To unset (discontinue) an editor option, type `no` before the option:

`:set no`*option* Return

To show only the changes you have made to the default options, type

```
:set
```

The editor options affect how `vi` works. For instance, when you first enter `vi`, you don't see line numbers preceding the lines of text. However, you can use the **number** option (abbreviated **nu**) to change this feature. Type either the full option name or its abbreviation **nu**:

```
:set number
```

or

```
:set nu
```

Each line will then be preceded by a line number, until you unset the number option or exit `vi`. To unset the number option, type:

```
:set nonumber
```

or

```
:set nonu
```

Table 6-9 describes some options, their abbreviated names, and default values. To determine the options most useful for your needs, you might open a practice file and experiment with editor options.

Table 6-9. Editor Options

Option	Abbrev.	Default	Effect When Set
`all`	~	~	Lists all editor options on the screen.
`autoindent`	`ai`	`noai`	Begins each new line of text in the same column as the previous line (useful for programmers).
`ignorecase`	`ic`	`noic`	Causes `vi` to ignore uppercase and lowercase during searches.
`number`	`nu`	`nonu`	Number each line of text.
`readonly`	~	`noreadonly`	Enables *write-protection* on the file you are editing. This ensures that you don't accidentally change or destroy the file's contents.

Table 6-9. Editor Options (continued)

Option	Abbrev.	Default	Effect When Set
showmatch	sm	nosm	Shows the opening parenthesis, brace, or bracket when you type the corresponding closing parenthesis, brace, or bracket. This option is useful when you are typing mathematical expressions or writing programs in a language that uses parentheses, braces, or brackets.
showmode	~	noshowmode	Displays a message like "INPUT MODE" or "REPLACE MODE" at the bottom of the screen whenever you are in either of these modes.
wrapmargin	wm	wm=0 (zero)	Changes the right margin. *n* equals the number of spaces in the right margin. For example, if you're using an 80-column terminal, then `:set wm=8` sets the right margin at column 72.

Although the non-default settings entered with this method will be lost each time you exit `vi`, the next section shows you how to make `vi` read your custom options automatically each time you open a file.

Making Your vi Environment Permanent

To avoid setting options or defining abbreviations or macros each time you enter `vi`, place all options and definitions you normally use into an `.exrc` file. (You can create this file, if it doesn't already exist). The `.exrc` file is a file in your home directory that contains all your changes to `vi` system defaults. Your changes to the `.exrc` file makes your customized `vi` environment permanent—until you decide to change the file again.

To change the default `.exrc` file, follow these steps:

1. Type `cd` at the HP-UX prompt to ensure that you're in your home directory; then use `vi` to create a `.exrc` file:

```
$ cd
$ vi .exrc
```

2. Type the options, word abbreviations, and macros you want to make permanent (don't precede the commands with a colon).
3. Type `:wq` to save the text and exit `vi`.

After creating the `.exrc` file, you can access it whenever you want to change your `vi` environment. Any of the editor options discussed in the previous section can be placed in this file for `vi` to read automatically each time you enter `vi`.

An Example of Changing Your .exrc File

In the previous section, "Using Options to Change Your vi Environment," you saw examples of some of the options which will change the overall behavior of `vi`. `wrapmargin` and `showmode` are two of the useful options which were shown in Table 6-9. You can also make `vi` recognize short forms of commonly used expressions by using `ab` to define an abbreviation.

Suppose you include the following options and abbreviations into your `.exrc` file:

```
set wm=8
set showmode
ab eecs Electrical Engineering and Computer Sciences
~
~
:
```

Then you've changed your `vi` environment so that each time you enter `vi` you can expect the following:

- The right margin automatically contains eight spaces (changing the default from zero), and a carriage return will happen after approximately 72 spaces.
- The lower right part of the screen will show "INPUT MODE" when you are in any of the text insert modes.
- The abbreviation *eecs* will automatically expand into the words *Electrical Engineering and Computer Sciences* whenever you enter *eecs*.

To change your `.exrc` again, use `vi` to edit as needed for your special needs. When you've finished changing the `.exrc` file, use the `:wq` command to save the changes.

For Additional Information

To obtain more details about how to use the `vi` screen editor, see *The Ultimate Guide to vi* and `vi` in Section 1 of the *HP-UX Reference*.

Chapter Command Summary

Table 6-10.

To Do This ...	Type This ...
Enter `vi`	`vi` *filename*
Make sure you're in command mode	Press ESC
Insert text before the cursor position	Press i
Insert text after the cursor position	Press a
Find out what file you are in	Press CTRL-g
Show whether you are in command or insert mode	`:set showmode`
Write (save) the current file.	`:w`
Write the current file to *file_name*	`:w` *file_name*
Overwrite the contents of *file_name* with the current file.	`:w!` *file_name*
Write lines *x* through *y* of the current file to *file_name*.	`:`*x,y* `w` *file_name where x,y represents specific line numbers or place markers.*
Copy a line into a buffer	`yy`
Put copied or deleted text line after cursor line	`p`
Read in another file after cursor line	`:r` *filename* Return
Find *text*	*/ text*
Run an HP-UX command while in `vi`	`:!`*command_name*
Print the current file	`:!lp %`
Print a file	`:!lp` *filename*
Write the current file and quit (exit) `vi`.	`:wq`
Quit `vi` *without* saving any changes.	`:q!`
Join lines of text	`J` at end of line
Set line length (`wrapmargin`)	`:set wrapmargin=n`

7

Sending and Receiving Mail

If you are on a multi-user system, you can send mail messages to other users on your system using the `elm` mailer. If your system is connected to a network, such as a local area network (LAN), you may also be able to send mail messages to users on other systems.

This chapter will help you in doing the following basic tasks in `elm`:

- Reading your mail
- Sending mail to users on your system
- Sending mail to users on other systems
- Deleting mail messages
- Saving mail messages
- Customizing `elm`

For further information on using `elm` and other mail systems in HP-UX, see *Mail Systems: Users Guide*

Getting Started with the Elm Mailer

To start the `elm` mailer, enter

```
$ elm
```

A display similar to the following appears:

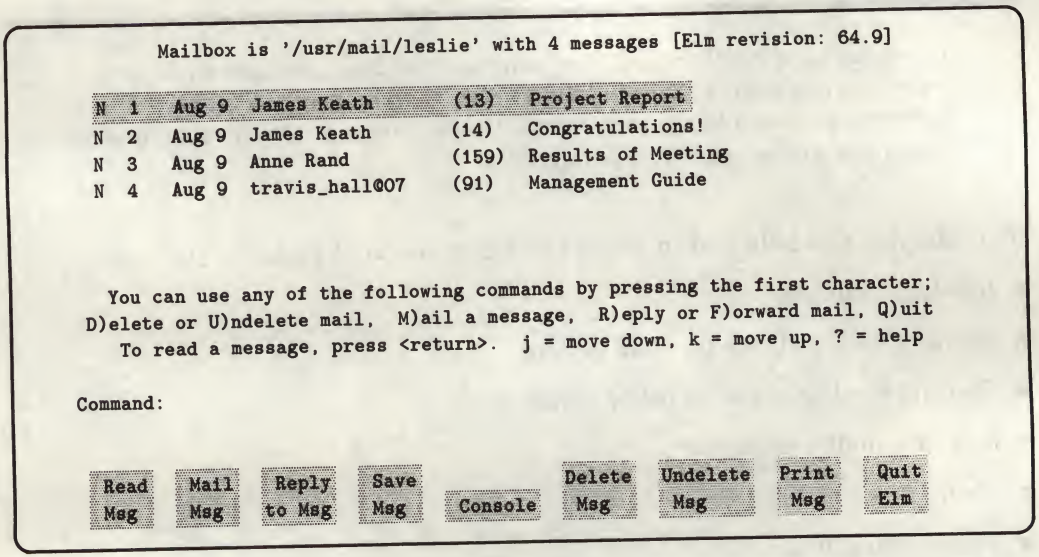


Figure 7-1.

7

You can enter an `elm` command in either of two ways:

- Type the first letter (uppercase or lowercase) of the command.
- Press the softkey that corresponds to the command. (You can execute the command menu choices, `Read Msg`, `Mail Msg`, etc. that appear at the bottom of the screen by pressing softkeys f1 through f8 at the top of your keyboard.)

The examples in this guide use the first method: typing the first letter of the command.

Leaving Elm

When you're in `elm`, you can leave it by typing the response to `Command`. If you have mail, `elm` will respond in the following way:

```
Command: q          Keep mail in incoming mailbox ? (y/n) y
```

You can either respond `y` to the prompt or press Return which also enters a `y` for your response.

Any messages in the incoming mailbox remain there, and the shell prompt returns. If you answer `n` to the prompt, messages are stored in an alternate mailbox, called `mbox`, in your home directory.

Setting Your Elm Environment: the .elm Directory

Like many programs in HP-UX, `elm` will automatically refer to a `~/.elm/elmrc` file for how to configure your version of `elm`. This will happen each time you start `elm`. If your home directory does not have the `.elm` directory with the `elmrc` file, you should create them:

```
$ mkdir .elm
$ cd .elm
$ vi elmrc
```

The `elmrc` file can contain different kinds of "customization" information. For example, it will tell `elm` how to sort your message lists: in alphabetical order, by the order in which they have come in, or in some other order. It can tell `elm` which type of editor you would like to use (if other than the one determined in your .profile). The section, "Customizing Elm," will give you more details on how `elm` will set up your `elmrc` file for you.

While you're in the `.elm` directory, you should also create a directory to store the names of users you commonly send mail to. These lists can then be identified and referred to by `elm` as one-word "aliases." The use of aliases makes group mailings much simpler.

For example, the following alias entry will enable you to mail to five different people with one "name" (some include an address for other systems):

```
call =  anne, pat, michael, don@ulpcu3, paul@ulpcu3,
```

Reading Your Mail

To read your mail, start the elm mailer. If you *do* have mail, `elm` displays a list of mail messages. You can read the *current* message or pick a specific message to read.

To determine whether you have any mail, type:

```
$ elm
```

The `elm` screen appears. If you have messages, `elm` lists the message information in a display similar to the following:

```
Mailbox is '/usr/mail/leslie' with 4 messages [Elm revision: 64.9]

N  1   Aug 9  James Keath        (14)   Project Report
N  2   Aug 9  James Keath        (14)   Congratulations!
N  3   Aug 9  Anne Rand          (159)  Results of Meeting
N  4   Aug 9  travis_hall@07     (91)   Management Guide
```

Figure 7-2.

To read the *current* message (that is, the message highlighted in inverse video) press Return. (On some systems the current message may be indicated by a > to the left of the message.)

7

Caution Problems with system behavior may result if you attempt to use one of the other HP-UX mailers, such as `mailx`, while you are also working in `elm`.

The following example shows the output from reading message 1.

```
Message 1/4 from James Keath

Subject:  Project Report
To: leslie
Date: Mon, 26 Nov 90 16:26:45 MDT
Cc:

The project report is in the mail.
Hope you enjoy it!

Best regards,

Jim
```

To return to the `elm` main screen, press Return.

Moving Through the Header Information

You can select a message to read as follows:

- Type `j` to advance to the next message, `k` to move to the previous one, just as you would move the cursor in `vi`. Press Return to read the message which you select.
- To jump to a specific message in the header, type the number of the message and press Return.

Displaying Message Headers

Only ten message headers appear on the screen at one time. If you have more than ten messages you can display them as follows:

- To see the next page of message headers, press `+`.
- To see the previous page, press `-`.
- To move to the first message in the list, press `=`.
- To move to the last message in the list, press `*`.

Sending Mail to Users on Your System

One of the easiest ways to learn how to send a mail message is to send one to yourself. If you're not already in `elm`, enter:

```
$ elm
```

To mail a message, type the response to `Command:`

```
Command: m
```

`elm` responds with a prompt requesting the *mail address* of the recipient.

```
Send the message to: leslie      Enter your own user name.
```

`elm` then responds with a *subject line* prompt:

```
Subject of message:      Type the subject line for the message.
```

For the message we're about to type, you might enter a subject line like:

```
Subject of Message: Important Message to Myself
```

After entering the subject line, press Return. `elm` responds with a prompt for the carbon copies:

```
Copies To:   Since you're sending a message to yourself,
             you don't want to send any additional copies
             so press Return.
```

7

On most systems, `elm` brings up the `vi` editor. (Some systems may be configured to bring up a different editor).

If you're in `vi`, press `i` to enter insert mode, and then begin typing the message. At the end of each line, press Return.

```
This is a mail message sent to myself.
Does it work?
We'll soon see.

Goodbye,
Leslie
```

To exit the `vi` editor and save your message, press esc and then :

```
:wq
```

For more information on the vi editor, see Chapter 6 in this manual.

After you exit the editor, you'll see the following message on the screen.

```
Please choose one of the following options by the first character:
E)dit message, edit H)eaders, S)end it, or F)orget it. s
```

To mail your message, type:

```
s
```

After you've sent the mail message, the elm main screen reappears and displays the message "Mail Sent." It might take a few minutes for the system to deliver the message.

Sending a Message to Multiple Recipients

To send a message to multiple users, specify each user's name next to the elm prompt for a recipient.

```
Send the message to:  mike leslie tj
```

Sending Mail to Users on Other Systems

If your system is connected to other systems over a LAN (local area network), UUCP or other networking facility that supports elm, you can send mail to users on other systems.

Node Names

Every system connected over a network has a unique **node name** (also known as a **host name**). When sending mail to each other, systems must know each others' node names. Figure 7-3 shows an example LAN with four systems connected. The node names are `research`, `develop`, `market`, and `sell`.

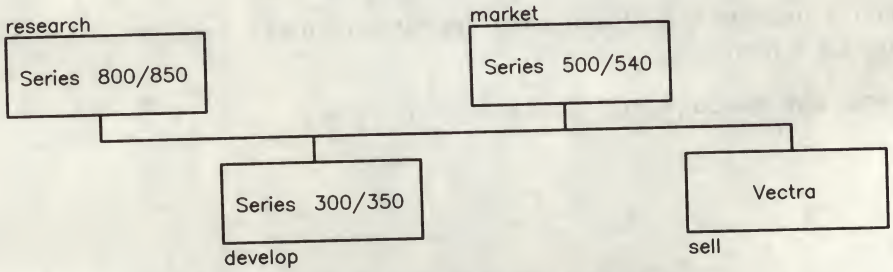


Figure 7-3. Example LAN and Node Names

7

To determine your system's node name, use the `hostname` command.

```
$hostname
market          Your hostname is displayed.
```

Mail Syntax when Mailing to Other Systems

The general syntax used when mailing to a user on another system is either:

node!user *UUCP format*

or

user@node *ARPA/Berkeley format*

In the above example, *node* is the node name of the system the person is on, and *user* is the person's unique user-name. Which syntax you use depends on whether your system uses UUCP or ARPA/Berkeley network services. Your system administrator will be able to tell you which syntax to use on your system.

Some Example Mail Addresses

Suppose you want to send mail to user `john`, who uses the system named `sell`. Then, in response to the `elm` prompt, you would use one of the following mail addresses:

```
sell!john
```

or

```
john@sell
```

To send mail to *arnie* on your system, *john* on the `sell` system, and *leopold* on the `research` system, use one of the following addresses:

```
arnie  sell!john  research!leopold
```

or

```
arnie  john@sell  leopold@research
```

Deleting Mail Messages

After you've read your mail messages, you may want to delete them. You can mark messages for deletion using the `d` command. When you execute either the `$` command, or if you quit elm, you will delete the marked messages.

Marking Messages to be Deleted

To delete a mail message, press `d` while the message is the current message. A `D` appears to the left of the message to show that it is marked for deletion.

You can mark additional messages for deletion by making each message the current message and then pressing `D`. *Remember, to move to the next message, press* `j`, *to move to the previous message, press* `k`. *You can also use the arrow keys,* ▼, ▲.

The following screen shows two messages marked for deletion.

```
   Mailbox is '/usr/mail/leslie' with 4 messages. [Elm revision: X.X]


D  1   Apr 3   Leslie    (6)    Important Message to Myself
   2   Apr 3   Robert    (24)   Meeting Tommorrow
   3   Apr 2   Lynn      (154)  More Software Requests
D  4   Apr 2   Patrick   (78)   Hi there
```

Delete the marked messages at the `command:` prompt in one of two ways:

- To delete the marked messages *and* quit elm, enter `q`. `elm` will ask you to confirm this action.
- To delete the marked messages *without* quitting elm, type `$`.

The next section tells you how you can store your messages in a different mailbox or save them to a file.

Saving a Mail Message to a File

When you `quit` `elm` you have the option of keeping your messages in the incoming mailbox (where your new messages arrive), or storing them in another mailbox (the default is *homedirectory*`/mbox`). While in `elm` you can also save messages to a designated file.

To save the current message to a file, at the `elm` command prompt, type:

```
Command: s
```

The following prompt appears;

```
Command: Save Message
File message in: =/username
```

If you press (Return), the message is saved in a file named with the sender's *username* in the `Mail` directory in your home directory. The default mail option is set so that the equal sign (=) is shorthand for *homedirectory*`/Mail`. If the `Mail` directory doesn't already exist, you need to create it.

If you want to save the message in another file, enter the name of the file. For example:

```
Command: Save Message
File message in:=/oldnews
```

7 Supposing that Leslie is the user, the current message is saved in the file `oldnews` in the `/user/leslie/Mail` directory. If the file already exists, the message will be appended to the contents of the file. If there is no existing `oldnews` file, one will be created.

After you save a message in a file, the message is marked with a `D` for deletion.

Two Ways to View Saved Messages

- The most common way to view saved messages is to change your mailbox and view them as you do in the incoming mailbox.

  For example, assume that you have saved in the file `oldnews` two messages titled "More Software Requests" and "Meeting Tomorrow." To view these messages in the `oldnews` mailbox, use the `change` mailbox command:

  ```
  Command: c
  ```

  One or more prompts appear asking you whether you want to keep mail in your incoming mailbox or delete it. Answer `y` or `n` as you choose.

  Then, `elm` prompts you for the name of another mailbox. After the "=" (which stands for "*homedirectory*`/Mail`") enter the name of the file, `oldnews`:

  ```
  Name of new mailbox:=/oldnews
  ```

  The mailbox changes to `=/oldnews`.

```
Mailbox is '=/oldnews' with 2 messages. [Elm revision: X.X]


1   Apr 3   Robert    (24)   Meeting Tommorrow
2   Apr 2   Lynn      (154)  More Software Requests
```

- A second way to view messages saved in a file is to use a shell command such as `more`. If you are in `elm` and want to access a shell command like `more` without leaving `elm`, enter "!" at the `Command:` prompt. For example, to examine the contents of the `oldnews` file:

  ```
  Command: !

  Shell Command: more /users/leslie/Mail/oldnews
  ```

  Press Return to return to `elm`.

Customizing Elm

The `elm` mailer has different options you can set to make it more convenient for you to use. Among features you can change are the menus that appear on the screen, the printer your mail is sent to, and the order in which your mail is listed in your mailbox. These are entered automatically in the `.elm/elmrc` file which you set up previously in your home directory.

Bringing Up the Option Menu

To bring up the option menu, press `o` at the `elm` command prompt:

```
Command: o
```

You'll see a menu similar to the following:

```
C)alendar file          : /users/leslie/calendar
D)isplay mail using     : builtin
E)ditor                 : /usr/bin/vi
F)older directory       : /users/leslie/Mail
S)orting Criteria       : Date Mail Sent
O)utbound mail saved    : /users/leslie/mbox
P)rint mail using       : pr %s | lp
Y)our full name         : Leslie Pendergrast

A)rrow cursor           : OFF
M)enu display           : ON

U)ser level             : 0 (for beginning user)
N)ames only             : OFF
T)abs to spaces         : OFF

Select first letter of Option line, '>' to Save, or R)eturn

Command:
```

Figure 7-4. The Options Menu

This guide does not describe all of the options in the option menu. Rather, it gives you an example of changing an option; in this instance how to change the order in which your mail is listed.

Changing the Order of Your Mail Messages

To change the order in which your mail messages are listed in your mailbox, press `s` (the first letter of `S)orting criteria`) in the alias menu.

```
Command: s
```

You'll see a message indicating how messages are currently sorted. For example:

```
This sort will order most-recently-sent to least-recently-sent
```

To see different choices for sorting your messages, press space bar. When you see the method you want, press Return.

For instance, when you see:

```
This sort will order by sender name
```

Press Return, then press > to save the change.

The change is entered in the `elmrc` file in the `.elm` directory that you have set up previously in your home directory.

To return to your mailbox, press Return again.

The messages in your mailbox will now appear in alphabetical order by sender name.

To get information about a specific option in the option menu, type `?` and then type the first letter of the option.

For More Information ...

For more information on the `options` command and the `elmrc` file see *Mail Systems: User's Guide* and the *elm*(1) entry in the *HP-UX Reference*.

Chapter Command Summary

Use the following commands from the Options Menu. Other commands are available from the Alias Menu.

Table 7-1.

To do this ...	Use this elm command ...
Delete the messages marked for deletion without quitting elm. ("Resync")	$
Get help on elm commands.	?
Allows you to send a command to the shell without leaving elm.	!
Set up mail aliases.	a
Change the mailbox.	c
Mark messages for deletion.	d
Forward the current message to another user.	f
Send a group reply to all recipients of the original message.	g
Move the message pointer to the next message (below).	j
Move the message pointer to the previous message (above).	k
Send mail to a specified user or users.	m
Allows you to alter the setting of different mail parameters, including the sorting method for messages, the destination of printed messages, the type of menus displayed, and so on.	o

Table 7-1. (continued)

To do this ...	Use this `elm` command ...
Print messages. (You can change the destination of printed messages using the `o` command listed above.)	`p`
Quit `elm` with the option of changing the contents of the mailbox.	`q`
Reply to the author of the current message.	`r`
Save a message to a file.	`s`
Exit `elm` without making any changes.	`x`

To see a summary of all of the commands you can use from `elm`, type "?" at the `elm` command prompt.

You can abbreviate every mail command (except `help`) merely by specifying the first letter. For example, you can abbreviate the `delete` command with a single `d`. To abbreviate `help`, use a question mark (`?`).

For More Information ...

For more information on the `elm` commands see the following entries in Section 1 of the *HP-UX Reference*:

elm
readmail
newmail
elmalias

8

Searching and Sorting

In addition to the `:/` command for use within `vi`, there are many valuable search and file-manipulation tools in HP-UX that work from a shell.

Specifically this chapter discusses:

- Searching for files using `find`.
- Searching for text patterns using `grep`.
- Sorting files using `sort`.

For more information on any of commands discussed in this chapter, see the *HP-UX Reference* entries for `find`, `grep`, and `sort`, or refer to the appropriate man page on-line.

Searching for Files using find

You can use the `find` command to search through a directory and its subdirectories for files meeting certain criteria. You can then execute a command on the files you've found. To display the output of `find` on the screen, you must use the `-print` option.

Finding Files that Match a Pattern

Although the syntax of `find` can be complex, it may help you in using HP-UX in a more productive way, simply by applying the following examples.

Suppose you want to display all files in the current directory and its subdirectories that begin with d. Enter:

```
$ find . -name 'd*' -print
```

The dot (.) causes `find` to search the current directory and its subdirectories. The `-name` option followed by a filename or a filename pattern (in this case `d*`) tells `find` to search for all filenames that match that pattern. In this example, `find` will look for all file names beginning with `d`.

Note that:

- `d*` is enclosed by single quotes `'d*'`. If you use a file name pattern in the `find` command, you must quote it so that the shell will interpret it correctly. The `-print` option displays the output on the screen.
- The order of `-name 'd*'` and `-print` is important. If `-print` were first, `find` would print *all* names.

Finding Files that are Newer than a Certain File

8

Suppose you want to display all files modified after a certain file. To display all files newer than `myfile` in the `/users/leslie` directory and its subdirectories, enter:

```
$ find /users/leslie -newer myfile -print
```

This example can be read as follows: find in directory `/users/leslie` and its subdirectories all files modified after `myfile` and display the output on

the screen. (To find out the date and time a file was last modified, use the `ll` command.)

Running Commands on Files

You can execute commands on files located with the `find` command. Let's say you want to remove all files with a `.tmp` extension in the current directory and its subdirectories. Enter:

```
$ find . -name '*.tmp' -print -exec rm {} \;
```

This example finds and displays on the screen all files in the current directory and its subdirectories that end in `.tmp`, then deletes these files. The `-exec` option causes the following command (`rm`) to be executed. The brackets { } represent the files found with the `find` command. The semi-colon that ends the command string is escaped with a backslash (\;).

Searching for Text Patterns Using grep

You can use the grep command to search for a text pattern within a file or to display the names of files that contain a specified text pattern. This command is useful when you want to search for information in files or directories.

Overview of the grep Command

The grep command ("global regular expression print") looks at each line of one or more files for a text string that matches a specified pattern. When it finds a matching text string, it displays the line in which the matching string is found.

Searching a File for a Text String

Suppose you have a mailing list called mailist with the contents shown below:

```
Smith, Joe        2345 Pine St.      Santa Clara, CA
Walsen, Stacy     493  Winkle Ave.   San Jose, CA
Diaz, Robert      6789 Pine St.      Santa Clara, CA
Wang, Michael     1832  Jackson St.  Santa Clara, CA
```

If you want to extract the addresses of all the people on Pine St. Enter:

```
$ grep Pine mailist
```

The grep command lists all lines in mailist that contain the string Pine. The output is:

```
Smith, Joe        2345 Pine St.      Santa Clara, CA
Diaz, Robert      6789 Pine St.      Santa Clara, CA
```

8

Searching Multiple Files

The grep command can be useful in other ways. Sometimes, you want to find information, but you don't know or can't remember in which file it's located.

Suppose you have three mailing lists, and can't remember which contains Stacey Walsen's address, enter:

```
$ grep 'Walsen, Stacey' mailist mailist2 mailist3
mailist: Walsen, Stacy    493  Winkle Ave.  San Jose, CA
```

The grep command displays the line containing Stacey's address and the file in which it was found. Note that because it contains a space, the string must be surrounded by single quotes ('Stacey, Walsen'). If grep had found other instances of the specified text string, it would list each instance that it found.

If you wanted to search the entire current directory for this information, you could simply enter:

```
$ grep 'Walsen, Stacey' *
```

Ordering Files Using sort

You can use the `sort` command to order the contents of files, sorting alphabetically, numerically, or by different fields.

Overview

The `sort` command displays sorted file contents on a line-by-line basis. It compares the first characters in each line; if they are the same, it compares the next two characters, and so on through the end of each line.

The `sort` command also recognizes different sortable fields. Fields are separated from each other by spaces or tabs. If there is more than one leading space before a field, `sort` will count each space as a sortable character.

Displaying Sorted File Contents in Alphabetical Order

Here is an example file called `list`, containing the names and telephones numbers of the members of a tennis club and the number of tickets they have sold for a benefit match. Each line in the file has four fields: the first name, the last name, the number of tickets sold, and the phone number. The fields are separated by spaces.

If you want to practice using the `sort` command, type in the `list` file using the `cat` command as shown in the following list:

```
$ cat > list
Nancy Smith      4    467-2345
Jeff Bettleman   8    438-7689
Jeff Plimpton    13   729-8965
Joyce Smith      6    245-1342
```

8

To display the sorted contents of the file, enter:

```
$ sort list
Jeff Bettleman   8    438-7689
Jeff Plimpton    13   729-8965
Joyce Smith      6    245-1342
Nancy Smith      4    467-2345
```

Note that the list is sorted alphabetically by *first* names. The `sort` command without any options will sort starting with the first field in each line (in this example the first name).

The `sort` command places the two "Jeffs" in the correct order, because after comparing the first names and finding them identical, it compares the second fields (last names).

Sorting Files by Different Fields

The next example shows how you can use `sort` to order the list by last names.

```
$ sort +1 list
Jeff Bettleman  8    438-7689
Jeff Plimpton   13   729-8965
Nancy Smith     4    467-2345
Joyce Smith     6    245-1342
```

The `+1` option causes `sort` to skip the first field of each line and start sorting at the second field (the last names). However, note that the first names Nancy and Joyce are not in alphabetical order.

The next example shows how to sort by last name and then by first name.

```
$ sort +1 -2 +0 list
Jeff Bettleman  8    438-7689
Jeff Plimpton   13   729-8965
Joyce Smith     6    245-1342
Nancy Smith     4    467-2345
```

The `+1` causes `sort` to skip the first field (first name) and sort by the second field (last name). The `-2` causes the `sort` command to stop after comparing the second fields. If any of the second fields are identical (in this case the Smiths), the `+0` instructs `sort` to make a second pass and sort from the beginning of the line (putting the first names in correct order).

Note that the `sort` command will only make a second pass at comparing lines if its first pass shows all of the examined fields in the lines to be identical. If `-2` were not specified in the above example, `sort` would show the third fields (telephone numbers) to differ and would never make a second pass to compare the first names.

Sorting in Numerical Order

Suppose you want to sort the file by the number of tickets sold. Enter:

```
$ sort -rnb +2 -3 list
Jeff Plimpton     13  729-8965
Jeff Bettleman    8   438-7689
Joyce Smith       6   245-1342
Nancy Smith       4   467-2345
```

There are several things to keep in mind when sorting in numerical order. First, `sort` will consider multiple leading blanks in its comparison. Since there are a variable numbers of blanks preceding the third field (ticket number), your results will be incorrect unless you use the `-b` (ignore leading blanks) option.

Also, the numbers will not be in the right order unless you use the `-n` (numeric) and `-r` (reverse) options. The `sort` command by default sorts from least to greatest numbers so you need the `-r` option to list the greatest number first. Also, by default `sort` works by comparing characters in a field in the order they appear. So without the `-n` option, 8 would come before 13, because 8 is larger than 1.

Chapter Command Summary

Table 8-1. General Commands

To do this ...	Type this ...
Sort the contents of a file and display the results to standard output	`sort` *options filename*
Find all filenames in the current directory and subdirectories beginning with "x"	`find . -name 'x*' -print`
Find the word "Jim" in any file in the current directory	`grep Jim *`
Find the words "Joe Smith" in any file in the current directory	`grep 'Joe Smith' *`

Table 8-2. Common sort Options

`sort` option ...	Does this ...
`-b`	(disregard leading blanks) Blanks, which can be either spaces or tabs, delimit sortable fields. Without the `-b` option, `sort` considers all blanks preceding a field to be part of that field, and will use them in sort comparisons.
`-d`	(dictionary) disregards all characters that are not alphanumeric or blanks. In particular, this option disregards control characters or punctuation.
`-f`	(fold) doesn't consider any difference between upper and lower case.
`-n`	(numerical) sorts in numerical order. Plus and minus signs are considered as plus and minus signs, and dots (.) are considered as decimal points.
`-r`	(reverse) reverses sort order (i.e., z-a).
`-u`	(unique) removes duplicate lines from a sorted file.

9

Using Your Shell Environment

HP-UX gives you your choice of several different shell-types which you can run. This chapter discusses the Bourne, Korn, and Key Shells. Information on the C Shell can be found in the Appendix. Each of these shells has different characteristics, and you can increase the speed and efficiency with which you interact with HP-UX if you learn to use some of the built-in features of the shell of your choice.

With the Korn Shell and the Key Shell, you can edit your command line and recall, and re-enter, previous commands. Your shell environment can be "customized" using shell variables and login scripts. One of the variables, PATH, enables the shell to find the correct directories for commands and files.

This chapter also tells you how you can temporarily or permanently invoke another shell-type.

This chapter covers these topics:

- Shell features: determining and changing your shell.
- Editing your command line.
- Recalling previous commands.
- Setting the login environment.
- Using login scripts to set the system environment.
- Setting and referencing variables.
- Finding commands with search paths.
- Setting terminal characteristics.

Shell Features, Determining and Changing Your Shell

Using simple commands, you can determine which shell you are running, temporarily change your shell, or permanently change your shell. Table 9-1 compares the features of the Bourne, Korn, Key, and C Shells. (The C Shell is discussed in detail in the Appendix.) Refer to Table 3-1 for a listing of both the file name for each shell and the default system prompt.

Below are listed some of the features which may help you make a decision on which shell would be best for the kind of work you are doing:

Table 9-1. Comparison of Shell Features

Features	Description	Bourne	Korn Key	C
Command history	A feature allowing commands to be stored in a buffer, then modified and reused.	No	Yes	Yes
Line editing	The ability to modify the current or previous command lines with a text editor.	No	Yes	No
File name completion	The ability to automatically finish typing file names in command lines.	No	Yes	Yes
`alias` command	A feature allowing users to rename commands, automatically include command options, or abbreviate long command lines.	No	Yes	Yes
Restricted shells	A security feature providing a controlled environment with limited capabilities.	Yes	Yes	No
Job control	Tools for tracking and accessing processes that run in the background. *See Shells: User's Guide*	No	Yes	Yes

Determining Your Login Shell

As you found in Chapter 1, the command `echo $SHELL` displays the file name of the shell you entered when you logged in.

```
$ echo $SHELL
/bin/sh
$ _
```

The `echo` command displays the contents or value of a variable named `SHELL`. The `SHELL` variable contains the name of the file that contains the shell program that you are running. In this case it is `/bin/sh`, the file that contains the code for the Bourne Shell.

Table 9-2 lists both the file name that displays for each shell and the default system prompt.

Table 9-2. Shell File Names and Default Prompts

Shell	File Name	Prompt
Bourne	/bin/sh	$
Korn	/bin/ksh	$
Key	/usr/bin/keysh	$
C (See Appendix)	/bin/csh	%
Restricted Bourne	/bin/rsh	$
Restricted Korn	/bin/rksh	$

Temporarily Changing Your Shell

Unless you are in a restricted shell, you can temporarily change your shell by using this command:

shell_name

where *shell_name* is the name of the shell (for example, `sh`, or `ksh`). Temporarily changing your shell lets you experiment in other shells. By typing the name of the shell you want to run, you *invoke* (enter) that shell, and the correct prompt is displayed. After experimenting in the new shell, return to your original shell by typing either `exit` or CTRL-D.

The following example begins in the Bourne Shell, enters the Korn Shell, and returns to the Bourne Shell:

```
$ ksh                                  Enter Korn Shell.
$ ps                                   Execute the ps command.
   PID TTY      TIME COMMAND
  6009 tty01    0:00 ksh               Notice that both the Korn Shell and
  5784 tty01    0:00 sh                Bourne Shell processes are running.
  6010 tty01    0:00 ps
$ exit                                 Exit Korn Shell.
$ _                                    Bourne Shell returns.
```

Permanently Changing Your Shell

To permanently change your *login shell* (the default shell you get when you log in), use the `chsh` change shell) command:

`chsh` *username full_shell_name*

where *username* is your user name and *shell_path_name* is the full path name (e.g., `/bin/ksh`) of the shell you want as your default. After you use the `chsh` command, you must log out and log in again for the change to take effect. For example, if `terry` changes the default login shell to the Korn Shell, the command reads:

```
$ chsh terry /bin/ksh
$ _
```

Editing Your Command Line

In the Korn Shell and the Key Shell, you can correct errors and make changes in the command line you are entering. You can make these changes with the *line-editing commands* or edit keys.

It can be very frustrating to type in a long command line, only to discover that there is an error in the first part of the line. In the Korn Shell, and in the Key Shell, you can correct such errors by using the line-editing commands. The built-in line editor allows you to move the cursor to the point in the line where you wish to make a change. There you can enter the correction, and execute the command. If you type in a line containing an error and execute it, it is also possible to recall that command, edit it and execute it again. (See "Recalling Previous Commands," later in this chapter.)

Using a Line-Editing Command Set

In Chapter 6 in this manual, you learned how to use `vi` screen editor with text files. The `vi` editor is also used to edit command lines.

To enter the `vi` line-editor mode while in the Korn or Key Shell, press (ESC) to change from the usual "typing mode" into "edit mode." to move the cursor to a new location or to delete characters. You then change back to "typing mode" by entering the `vi` commands `i` or `a` to insert or append text. In Key Shell, you can also use the arrow and editing keys on your terminal.

Note that you can use the line-editing features *only* in the Korn Shell and the Key Shell. If you are in another shell, change to Korn Shell before proceeding further.

```
$ /bin/ksh
```

Table 9-3 contains a partial list of the editing commands available.

Table 9-3. vi Line-editing Commands in Korn Shell

Desired action	vi command
Move back one character	h
Move forward one character	l
Move back one word	b
Move forward one word	w
Move to the beginning of the line	^
Move to the end of the line	$
Delete the character under the cursor	x

The editor command set you will use is governed by the setting of the EDITOR variable. Some possibilities are `vi` or `emacs`. Setting the EDITOR variable also depends on the VISUAL variable being defined. If you decide to use the `vi` editor on a temporary basis, set it in this way:

```
$ set -o vi        "set minus letter-o vi"
$ _
```

To turn off the `vi` editing mode, type:

```
set +o vi
```

To set the EDITOR variable automatically each time you log in, see Chapter 11, "Customizing Login Scripts," in this manual.

Executing vi Line-Editing Commands

To execute the `vi` editor commands, press ESC (to enter the command mode), then enter `vi` commands for the desired actions. To return to "typing mode," press i (to insert text at the current position of the cursor) or a (to append text at the point immediately to the right of the cursor).

An Example of Line Editing with the vi Command Set

Activate the `vi` command set (if it is not already activated at login by your login script):

```
$ set -o vi
$ _
```

Type this line *without* pressing (Return):

```
$ ll /dve | grep '^d' | more_
```

The second element should have been `/dev`. Correct the error by following these steps:

1. Press (ESC). The cursor moves back one space (beneath the `e` in `more`). The line editor is now in "command mode."

   ```
   $ ll /dve | grep '^d' | more
   ```

2. Press (h) repeatedly to move the cursor to the beginning of the line.

   ```
   $ ll /dve | grep '^d' | more
   ```

3. Press (L) five times. The cursor moves beneath the `v` in `/dve`.

   ```
   $ ll /dve | grep '^d' | more
   ```

4. Press (X). The character `v` disappears, and the rest of the line shifts one space to the left to fill in. The cursor is now under the `e` in `/de`.

   ```
   $ ll /de | grep '^d' | more
   ```

5. Press (A). The cursor moves one space to the right. The line editor is now ready to "append" text to the line.

   ```
   $ ll /de_| grep '^d' | more
   ```

6. Press (V). The character `v` is inserted after `/de`, completing the correction.

   ```
   $ ll /dev_| grep '^d' | more
   ```

7. Press (Return) to execute the command line.

For More Information ...

The `vi` line-editing command set works in a way that is very similar to the full `vi` editor. See Chapter 6 in this manual for an introduction to `vi`.

You can also choose to use the `emacs` editor, in which case you can find information in the *emacs* section of the *ksh*(1) reference in the *HP-UX Reference*.

Recalling Previous Commands

The Korn and Key Shells store the commands you execute in a **command history**. You can retrieve these commands, modify them, and re-execute them.

The Korn Shell and the Key Shell have a built-in "memory" of the command lines you type in. These commands are stored in a special area called the **command history**. For information on the C Shell implementation of command history, see the Appendix of this manual.

The Korn Shell's Command History

First make sure you are in the Korn Shell. If you are not, type the following:

```
$ /bin/ksh
$ _
```

The cursor will remain the same, although the shell has changed.

Execute some commands, as a test. Then, to re-execute a previous command:

- Make sure you have set `vi` as the command line editor (enter `set -o vi` on the command line for the login session, or make the the appropriate entries in your `.profile` to set and export the `EDITOR` variable.)
- Press (ESC).
- Then press (k) repeatedly to scroll "up" to the previous command that you want.
- Or press (j) to scroll back "down" through the command history list.
- Once you have found the command you want, you can edit it just as if it were the current command.
- You can then execute whatever is on the command line by pressing (Return).

The Korn Shell "remembers" the last 128 command lines you typed in and can display all or any of them. For example, type in some commands:

```
$ date
Wed Mar 29 10:57:28 MST 1989
$ pwd
```

```
/users/terry
$ hostname
hpfcma
$ _
```

Now type in this command:

```
$ history -3

121     date
122     pwd
123     hostname
124     history -3
$ _
```

Notice that the Korn Shell displays the last three commands (`date`, `pwd`, and `hostname`) *and* the `history -3` command. You can increase the amount of the command history shown by using a larger negative number to follow `history`. For example, this will display the last 100 commands if there are 100 commands in the history:

```
$ history -100
```

If there are fewer than 100 commands in the history, the full contents of the history will be displayed.

Key Shell will also display command history, with the added option of allowing you to use the terminal arrow and editing keys (instead of `vi`) to scroll through the command history and edit commands. As with Korn Shell, once you have displayed the command line you want, you can execute it by pressing Return.

For More Information ...

For more details on the command history in the Korn Shell, see the relevant tutorials in the *Shells: User's Guide*. For more information on the Key Shell, see Chapter 4 in this manual.

You can also find information on the C Shell command history in the Appendix of this manual, and in the *Shells: User's Guide*. Briefer presentations are available in the *ksh*, *keysh*, and *csh* entries in the *HP-UX Reference*.

Setting the Login Environment

When you log in, your shell automatically defines a working **environment** for you that is uniquely identified from every other environment on the system. Your environment defines such characteristics as who you are, where you are working, and what processes you are running. Your shell maintains this environment until you log out. The characteristics of your environment are defined by values assigned to environment variables.

Your shell environment is analogous to an office environment. In the office, physical characteristics like lighting and temperature are similar for everyone. But many factors in your office environment are unique to you, such as your routine tasks and your individual workspace. Thus, your work environment is different from that of your co-workers—just as your shell environment is different from theirs.

The login Program

When you log in, HP-UX runs a program named `login`. This program starts your session using data stored in the `/etc/passwd` file, which contains one line for each system user. This file includes your user name, password (in encrypted form), home directory, and the shell to run when you log in. If `/etc/passwd` doesn't specify a shell, the Bourne Shell (`/bin/sh`) is selected.

The `login` program executes when you type your user name at the prompt (`login:`). Its tasks include the following:

- Display the `Password:` prompt (if you have a password).
- Verify your user name and password in the `/etc/passwd` file.
- Assign default or user-defined values to the shell environment.
- Start executing the shell process.

Environment Variables

The shell environment defines how HP-UX interacts with you. The environment's characteristics are defined by **environment variables**, which consist of a name and a value. For example, the directory in which you begin each session is your **home directory**; its environment variable is the variable

named HOME, and its value is assigned during the login process. Throughout this section, the value of HOME is equal to /users/terry.

The following table contains the environment variables set during the login process. They are available to all three shells.

Table 9-4. Environment Variables Set During the Login Process

Variable	Description	Typical Default Value
HOME	Defines the user's home directory; the default directory for the cd command (for example, /users/terry).	Assigned during login
LOGNAME	Contains the user name (for example, terry).	*username*
MAIL	Determines where the system looks for mail. Set based on the user name (for example, /usr/mail/terry).	/usr/mail/*username*
PATH	Sets the directories through which the system searches to find and execute commands.	/bin:/usr/bin: /usr/contrib/bin: /usr/local/bin
SHELL	Determines which shell to run. Set to the last field in the /etc/passwd file entry for the user logging in. If this field is not defined, the default value is used.	/bin/sh
TERM	Specifies the kind of terminal for which output is prepared.	hp
TZ	Provides the current time zone and difference from Greenwich Mean Time. Set to Mountain Standard Time by default; your system administrator should change the value if you are in another time zone. Set by the script /etc/profile.	MST7MDT

Using Login Scripts to Set the System Environment

During the login process, HP-UX prompts you for your user name and password (if applicable) before displaying a shell prompt. HP-UX also notes which shell you've selected to run, starts your shell process, and sets up your environment referring to **login scripts**. A login script is a file that lets you customize your environment.

A **login script** is a file containing commands that let you define your system environment. When you log in, default values are assigned to environment variables. Login scripts provide an automatic way to change the value of these variables every time you begin a session. If login scripts exist, they are executed by your shell before you get a shell prompt.

Two types of login scripts are used:

- A system script for all users of a particular shell on your system or HP-UX cluster.
- Local login scripts in your own home directory.

Typically, a system administrator maintains the system login scripts. (If there is no one in your group responsible for these tasks, refer to *Managing Clusters of HP 9000 Computers* and to *System Administration Tasks*). These scripts set up a default environment for everyone on that system. The Bourne Shell and Korn Shell use a system login script named `/etc/profile`.

You maintain the local login scripts in your home directory. The local scripts allow you to set up an environment specific to your needs. The Bourne Shell looks for one script: `.profile`. The Korn Shell uses two login scripts: `.profile` and the one referred to by the `ENV` variable (by convention, it is called `.kshrc`).

Why Use Login Scripts?

Login scripts provide a convenient way to set up the shell environment to suit individual needs. For example, the script can change the value of the search path used to find commands, change the shell prompt, set the terminal type, or simply cause the shell to greet you with a friendly message of your choosing.

Customizing your login script is not required, and the login script your system administrator provides should set up the most critical shell parameters. As you experiment with your shell's capabilities, your desire to be creative and customize the login script will probably increase. You may want to refer to the next chapter, "Customizing Login Scripts," for examples of how to customizing your own login scripts.

A Summary of Login Scripts

Table 9-5 summarizes the login scripts for each shell. All the scripts run when you first log in. For the Korn Shell, `$ENV` is run for each subshell and this variable may refer to other scripts.

A **subshell** is an entirely new shell that your current shell creates to run a program you have requested. For example, typing `ksh` at any shell prompt creates a Korn Shell subshell.

Table 9-5. System and Local Login Scripts

Shell	Program Name	System Login Script	Local Login Script
Bourne	/bin/sh	/etc/profile	$HOME/.profile
Korn	/bin/ksh	/etc/profile	$HOME/.profile $ENV
Key	/usr/bin/keysh	/etc/profile	$HOME/.profile $ENV

Setting and Referencing Variables

Your shell uses both environment variables and shell variables to define your environment. Your login shell uses **environment variables** and passes them to all processes and subshells that you create. **Shell variables** are known only to your current shell and are not passed to subshells.

The Bourne Shell and Korn Shell set variables using an assignment statement and an optional `export` command. In all shells, you refer to the *value* of a variable by placing a dollar sign (`$`) in front of the variable name.

To use variables, you will find out how to:

- **set** (assign a value to) a variable.
- **reference** (refer to) the variable's value.

Assigning Values to Variables

In the Bourne, Korn, and Key Shells, variables are assigned (or set). They can also be created, if necessary. Both tasks are done with an assignment statement:

name=value

The *name* is the variable name and *value* is the value assigned to the variable. No spaces are allowed between *name* and = or between = and *value*.

In the following example, which works with Bourne or Korn Shells, the shell prompt (`PS1`) is reset so that it reads: "`Ready ==>` ." If `PS1` is a shell variable, the subshell (created by typing `sh`) does not know the new value. If you export `PS1`, the value of `PS1` passes to the subshell:

```
$ PS1="Ready ==> "          Set shell variable PS1.
Ready ==> sh                Enter subshell.
$ exit                      Subshell has default prompt; exit returns to
                            original shell.
Ready ==> export PS1        Set environment variable with export.
Ready ==> sh                Enter subshell.
Ready ==> _                 Subshell knows the new value of PS1.
```

Referencing the Values of Variables (Parameter Substitution)

All three shells use **parameter substitution** for referencing the value of variables. Parameter substitution means that the variable's value is substituted for the variable name. Parameter substitution occurs when a dollar sign ($) is placed in front of the variable name.

Earlier in this guide, you learned to determine your login shell with the command `echo $SHELL`:

```
$ echo SHELL          Because $ is omitted, the word SHELL is echoed.
SHELL
$ echo $SHELL         The $ is included, so the value of SHELL is echoed.
/bin/sh
$ _
```

The `echo $SHELL` command uses parameter substitution. The shell substitutes the value of the environment variable named `SHELL` into the `echo` command because the dollar sign ($) precedes the variable name.

For More Information ...

To learn more about parameter substitution, refer to *sh*, *ksh*, *keysh*, or *csh* in the *HP-UX Reference* (section 1) or to the *Shells: User's Guide*

Finding Commands with Search Paths

When you type a command, HP-UX must be able to find the directory containing the command before it can run the command. The PATH environment variable contains a list of directories you want HP-UX to search when looking for commands. Your PATH should contain all the directories necessary to locate all the commands that you use. If necessary, PATH can be customized, so that non-standard directories are searched.

PATH Variable Format

The PATH variable is read from your `.profile` or `/etc/profile` login script. If you look at PATH in one of these files, you will see that it contains a list of directories to search, separated by colons. There should be no spaces surrounding the colons.

You can also use the `echo` command to determine the current value of PATH, as follows:

```
$ echo $PATH
/bin:/usr/bin
```

This line means that when you type a command, the shell first searches for the command in the `/bin` directory, then in the `/usr/bin` directory. If the command isn't found in either directory, the shell displays this message:

command_name: `Command not found.`

To determine the current value of your PATH variable, use the `echo` command as shown above. For example, suppose you run `echo` and get a response as follows:

```
$ echo $PATH
/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin
```

In this example, the shell searches through `/bin`, `/usr/bin`, `/usr/contrib/bin`, and `/usr/local/bin`—*in that order*—to find commands. The shell will execute the first instance of the command that it finds along this path.

Changing PATH

If the shell can't find a command that you know exists, you have two options:

1. You can type the full path name of the command. For example, if you wish to execute a command called `prog`, and it resides in the directory `/users/leslie/bin`, type this:

   ```
   /users/leslie/bin/prog
   ```

2. However, you can also change the value of the `PATH` variable to add the command path, `/usr/contrib/bin`. This may offer a better long-term solution if you use the command frequently.

Table 9-6 shows the path names of the most frequently used directories. You might want to add some (or all) of these directories to your `PATH`.

Table 9-6. Possible Directories to Include in PATH

Directory	What It Contains
`/bin`	Frequently used HP-UX commands.
`/etc`	Commands the system administrator uses.
`/usr/bin`	Additional HP-UX commands.
`/usr/contrib/bin`	Contributed programs not supported by Hewlett-Packard.
`/usr/local/bin`	Programs and commands written locally (at your location).
`$HOME/bin`	A directory you might create for your shell scripts and programs.

Caution Because of the potential security risk, you would not usually put your current directory (represented as . or the equivalent) as the first element in PATH. Leave the current directory out of your `PATH`, or include it only as the *last* element.

Remember that directories in PATH are searched in the order in which they appear (left to right). In general, put the most frequently used directories first in the path—unless two commands in the search path have the same name (for example, /bin/rm and $HOME/bin/rm). In this example, if you want the shell to find your version of rm first, put $HOME/bin before /bin in PATH.

The following example shows how to alter PATH to include $HOME/bin before any other directories, and to include the current directory as the last directory in the search path (this example assumes you're using the Bourne, Korn, or Key Shell):

```
$ echo $PATH
/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin
$ PATH=$HOME/bin:$PATH:.      Including . as the last element makes
$ echo $PATH                  the current directory the last one searched.

/users/terry/bin:/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin:.
$ _
```

Setting PATH as an Environment Variable

Normally, you set `PATH` as a environment variable, so it is set to the appropriate value when you log in. In the Bourne and Korn Shells, you can change `PATH` in the `.profile` script and export it. You will learn more about these scripts in Chapter 10, "Customizing Login Scripts," in this manual.

Chapter Command Summary

Table 9-8. Commands

To Do This ...	Type This ...
Determine what shell you're in	echo $SHELL
Temporarily change to Bourne Shell	sh
Temporarily change to Korn Shell	ksh
Temporarily change to C Shell (see Appendix)	csh
Permanently change to another shell	chsh *username shell_path_name* (then log out and log in again)
Set command-line editor	set -o *editor_name*
Edit your command line (once editor is set)	Press ESC; use `vi` commands to move cursor and enter text
Recall a previous command line	In `vi` mode, press ESC; press `k` (backwards) or `j` (forward) to move through command history file
Execute a previous command line	Press Return when desired command line is displayed
Set a variable value	*VARIABLE_NAME=variable_value*
Enter a subshell	sh
Display PATH setting	echo $PATH
Set terminal parameters	tset *options term_type*

10

Customizing Login Scripts

The examples in this chapter show HP-UX default login scripts and local login scripts that customize your shell environment. Login script examples are provided for the Bourne Shell and the Korn Shell.

This chapter covers the following topics:

- The types of customization used in login scripts and how to use the examples in this chapter.
- The default system login script for the Bourne and Korn Shells—`/etc/profile`.
- The default local login script for the Bourne and Korn Shells—`/etc/d.profile`.
- An example Bourne Shell `.profile` script.
- An example Korn Shell `.profile` script.
- An example Korn Shell `$ENV` (`.kshrc`) run-command login script for defining local shell variables.

Overview of Login Script Examples

By modifying your login scripts, you can customize your shell environment to meet your needs. This chapter describes default system login scripts that contain examples of setting the time zone variable and checking for mail, news, and message of the day. Most types of customization are included in at least one script. For example, all shells let you permit or deny messages to your terminal, but only the example Korn Shell `.profile` script illustrates the **mesg** command.

The scripts in this chapter are presented in two sets. In each set, the script commands are explained to help you understand what each script does. The first set presents scripts containing copies of the default scripts provided with HP-UX. The default login scripts are described in the sections that begin with "The Default.... " The second set presents scripts containing examples for customizing your local login scripts. The local, customized login scripts are described in the sections that begin with "An Example.... "

Ways to Customize Login Scripts

The following types of customization are commonly used in login scripts:

- Setting terminal characteristics.
- Setting search path and other global environment variables.
- Setting local shell variables.
- Setting default file permissions with **umask**.
- Setting up a logout script with **trap** (Bourne, Korn, and Key Shells only).
- Setting command aliases (Korn, Key, and C Shells only).
- Setting history variables (Korn, Key, and C Shells only).
- Displaying system status information and friendly messages.
- Checking for the existence of mail.
- Permitting or denying messages to your terminal with **mesg**.
- Setting the time zone variable.
- Checking for news and the message of the day.

Executing the System Login Script

If you use the Bourne Shell or Korn Shell, `/etc/profile` is your system login script. To view the contents of your system login script, use the `more` command.

When you log in, the system login script executes all its commands before the local login scripts execute.

If you know what the system login script does, you can avoid duplication in your local login script. For example, the default system login script checks for mail, so your local login script does not need to check for mail, too. However, if the system login script sets a different value for `PATH` than the value you need, then change `PATH` in your local login script.

Executing the Local Login Script

After the system login script executes, the local login scripts execute as follows:

- The Bourne Shell uses one local login script—the `.profile` script. All customization for this shell is done within the single login script, including both global environment variables and local shell variables.
- The Korn Shell uses two local login scripts. The `.profile` script executes after the system login script. If the `ENV` variable has been set to `$HOME/.kshrc`, the `.kshrc` script executes after the `.profile` script and each time you create a subshell.

In the Korn Shell, the `.profile` and `.login` scripts define global environment variables, terminal characteristics, and other parameters that apply only to the login shell. The `.kshrc` scripts define local shell variables and other parameters that apply to each subshell created.

The Default System Login Script for the Bourne and Korn Shells: /etc/profile

```
# @(#) $Revision: 66.4 $
# Default (example of) system-wide profile file (/bin/sh initialization).
# This should be kept to the bare minimum every user needs.
 trap "" 1 2 3                    # ignore HUP, INT, QUIT now.
 PATH=/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin  #default path.
 if [ -r /etc/src.sh ]
 then
    . /etc/src.sh                 # set the timezone
    unset SYSTEM_NAME
 else
     TZ=MST7MDT                   # change this for local time.
     export TZ
 fi
 if [ "$TERM" = "" ]        # if term is not set,
 then          #
  TERM=hp                         # default the terminal type
 fi
 export PATH TERM
# set erase to ^H
 stty erase ^H
# Set up shell environment:
 trap "echo logout" 0
# This is to meet legal requirements...
 cat /etc/copyright
 if [ -f /etc/motd ]
 then
  cat /etc/motd                   # message of the day.
 fi
 if [ -f /bin/mail ]
 then
  if mail -e                      # notify if mail.
  then echo "You have mail."
  fi
 fi
 if [ -f /usr/bin/news ]
 then news -n                             # notify if new news.
 fi
 if [ -r /tmp/changetape ]                # might wish to delete this:
 then echo "\007\nYou are the first to log in since backup:"
  echo "Please change the backup tape.\n"
  rm -f /tmp/changetape
 fi
 trap 1 2 3                           # leave defaults in user environment.
```

Task	Explanation
Ignore signals	The opening `trap` command prevents anyone from interrupting the execution of the login script because the following signals are ignored: `hangup` (1), `interrupt` (2), and `quit` (3). These signals are described in *signal*(5) in the *HP-UX Reference*.
Set `PATH`	The default `PATH` tells your shell to search `/bin`, `/usr/bin`, `/usr/contrib/bin`, and `/usr/local/bin`. Your current directory is not included.
Set time zone and system messages	`/etc/rc` runs a `set_parms` script the first time your system boots. It allows you to interactively enter your time zone and system i.d. information. That data is written to `/etc/src.sh`, which is subsequently read to set the TZ variable. If this script isn't found, then `.profile` sets the TZ variable, in this case, to Mountain Standard Time and exports it.
Set `TERM`	If the `TERM` variable is not set, this script sets it to `hp`.
Export global variables	This expression exports `PATH`, and `TERM` to subshells.
Set erase	This sets the erase character to CTRL-H.
Trap logout	This `trap` command sets a trap on exit. When the shell exits, the command "`echo logout`" is executed.
Display `copyright` file	The `cat` command displays the contents of `/etc/copyright`. This file contains copyright messages required by law.
Display `motd` file	If the `/etc/motd` ("Message of the Day") file exists as a regular file, the `cat` command displays its contents.
Check for mail	If the file `/bin/mail` exists, the `mail -e` command checks your mailbox. If you have mail, the message "`You have mail`" is displayed.
List news items	If the file `/usr/bin/news` exists, the `news -n` command is executed. This option displays the list of news files in `/usr/news`, rather than displaying the contents of the files.
Check for backup tape	If the file `/tmp/changetape` exists and is readable, then the message, "`Please change the backup tape`" is displayed. Check your site's backup procedures before changing the tape.
Reset signals	The final `trap` command resets the signals for hangup (1), interrupt (2), and quit (3). After the login script finishes executing, you are able to use hangup, interrupt, and quit again.

The Default Local Login Script for the Bourne and Korn Shell: /etc/d.profile

The Bourne Shell and Korn Shell use a local login script named `.profile` located in the home directory. Your system administrator or system installer gives you a default local login script based on the file `/etc/d.profile`. If you copy and edit the default login script yourself, make sure your copy is renamed `$HOME/.profile`, so that the system can recognize it.

The default local login script resets the value of `PATH`, sets terminal characteristics, sets up the shell environment, and sets shell variables.

```
# @(#) $Revision: 64.3 $

# Default user .profile file (/bin/sh initialization).

# Set up the terminal:
        eval ' tset -s -Q -m ':?hp' '
        stty erase "^H" kill "^U" intr "^C" eof "^D"
        stty hupcl ixon ixoff
        tabs

# Set up the search paths:
        PATH=/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin:.

# Set up the shell environment:
        set -u
        trap "echo 'logout'" 0

# Set up the shell variables:
        EDITOR=vi
        export EDITOR
```

Task	Explanation
Set terminal characteristics	The `tset` command generates the `TERM = (hp)` prompt during login. Based on the value you enter, the `TERM` environment variable is set and the terminal's characteristics are defined. If you press Return, `TERM` is set to `hp`.
	The `stty` command sets key definitions, options for modem connections, and tabs. Refer to *stty*(1) for more information.
Set `PATH`	The value of `PATH` is explicitly changed, meaning the entire value of the variable is reset. The new value adds the current directory to the end of `PATH`. If you attempt to execute a command that HP-UX cannot locate along any of the other paths, this value of `PATH` causes the shell to search for the command in the current directory.
Set shell environment	The `set -u` command prevents the shell from attempting to execute command lines with undefined shell variables. This command is a safety feature because, by default, the shell substitutes a null value for variables that aren't defined. For example, if a script uses a variable named `WORK` and you have not assigned a value to `WORK`, the script will not execute.
	The `trap` command sets a trap on exit of the shell. When the shell exits, the command "`echo logout`" is executed. (This `trap` command duplicates the one set in `/etc/profile`.)
Set shell variables	The shell variables `EDITOR` and `LANG` are set and exported as global variables.
	`EDITOR` is a Korn Shell variable that determines which line-editing command set the shell uses. To be certain that this variable is used to select the preferred line editor, make sure that the `VISUAL` variable is *not* set or that it is set to the path of a compatible screen editor.

An Example Bourne Shell .profile Script

The Bourne Shell uses a login script named `.profile` located in your home directory. From `.profile`, exported variables are passed to all subshells. Variables that are not exported are used only by the login shell. The following script sets terminal characteristics, global environment variables, and local shell variables used only by the login shell. The script also sets up a logout script and displays some system status information.

```
# Set terminal characteristics:
eval ' tset -s -Q -m ':hp2623a' '
tabs -T$TERM

# Set PATH:
PATH=$HOME/mybin:/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin:.

# Export global variables:
export PATH

# Set local shell variables:
PS1='$LOGNAME $ '
CDPATH=.:..:$HOME

# Set up logout script:
trap "echo logout; $HOME/.logout" 0

# Display system status information:
date
echo Currently `who | wc -l` users logged in.
```

Task	Explanation
Set terminal characteristics	This `tset` command sets and exports the `TERM` environment variable for an HP 2623A terminal. The `tset` command also initializes your terminal's characteristics. The `tabs` command with the `-T` sets the tabs to the default format for your terminal.
Set `PATH`	This example sets `PATH` explicitly to search `$HOME/mybin`, `/bin`, `/usr/bin`, `/usr/contrib/bin`, `/usr/local/bin`, and the current directory. A colon (`:`) separates each directory. The period at the end of `$PATH` specifies the current directory.
Export global variables	The `export` command allows shell variables to be passed to all commands that you execute. In this script, `PATH` is exported.
Set local shell variables	The Bourne Shell has several predefined shell variables. This script sets two: `PS1` and `CDPATH`. (These variables could also be exported to become global variables.) `PS1` is the primary shell prompt. Its default value is "`$ .`" In this script, `PS1` is changed to your username, followed by a dollar sign and a space. `CDPATH` sets the search path for the `cd` command. A colon (`:`) separates each directory. As set here, the `cd` command searches for the named directory in the current directory (`.`), the parent directory (`..`), and the home directory (`$HOME`).
Set up logout script	The Bourne Shell does not check for a `.logout` script automatically. The `trap` command sets a trap on exit of the shell. In this example, the `trap` echoes "`logout`" and executes a logout shell script when you log out.
Display system status information	The `date` command displays the current date and time. The `` `who | wc -l` `` "pipeline" counts the number of users logged in and displays the results in the echoed string.

An Example Korn Shell .profile Script

The Korn Shell distinguishes between **environment variables** and **shell variables**. Environment variables are used by your shell when logging in or creating subshells. Shell variables are local to your current shell and are not passed to subshells. Environment variables are defined in the `.profile` script, and shell variables are defined in a `.kshrc` script.

The following script sets terminal characteristics, global environment variables, and local shell variables, with the login shell. The script also turns off the ability to send messages to the terminal.

```
# Set terminal characteristics:
eval ' tset -s -Q -m ':3001' '
stty susp ^Z

# Set environment variables:
PATH=$HOME/mybin:$PATH
ENV=$HOME/.kshrc
PS1="'hostname' !$ "
CDPATH=.:..:$HOME

# Export global variables:
export PATH ENV PS1

# Set local shell variables:
MAILPATH=/usr/mail/terry:/usr/mail/kim%"Kim has mail"
MAILCHECK=300

# Deny messages to the terminal
mesg n
```

Task	Explanation
Set terminal characteristics	The `tset` command sets and exports the `TERM` environment variable for a low-resolution graphics display station. `tset` also initializes your terminal's characteristics. The `stty` command arranges for CTRL-Z to be the command for suspending jobs. This simplifies moving jobs into the background.
Set environment variables	This example sets `PATH` by adding to the existing value of `PATH` which is specified by `$PATH`. The `$HOME/mybin` directory is added as the first directory. With this search path, command names that you duplicate in your `mybin` directory will be used in place of the HP-UX version of the command. The `ENV` environment variable sets `$HOME/.kshrc` as the run-command login script. The `PS1` environment variable is the shell prompt. Its default value is "`$ .`" In this script, `PS1` contains the output from the `hostname` command, the command event number, and the dollar sign (`$`). `CDPATH` sets the search path for the `cd` command. A colon (`:`) separates each directory. As set here, the `cd` command searches for the named directory in the current directory (`.`), the parent directory (`..`), and the home directory (`$HOME`).
Export global variables	The `export` command allows shell variables to be passed to all commands that you execute. `PATH`, `ENV`, and `PS1` are exported in this example.
Set local shell variables	The Korn Shell has several pre-defined shell variables. `MAILPATH` and `MAILCHECK` are set in this example. The `MAILPATH` environment variable sets a search path for mail. In this example, the shell sends a message whenever mail arrives in `/usr/mail/terry` or `/usr/mail/kim`. For `terry`, the default message "`You have mail`" is displayed. For `kim`, the message "`Kim has mail`" is displayed. The `MAILCHECK` variable determines the frequency that the shell checks for mail. The default value is 600 seconds (10 minutes). In this example `MAILCHECK` is changed to 300 seconds.
Deny messages to terminal	The `mesg` command with the `n` option forbids messages written to your terminal via the `write` command.

An Example Korn Shell .kshrc Script

If the Korn Shell is your login shell, you can set up a run-command login script. This script (usually called `.kshrc`) defines local shell variables. Before creating a `.kshrc` script in your home directory, be sure that the environment variable `ENV` is defined and exported in your `.profile` script. This variable normally contains `$HOME/.kshrc`. When this is done, the `.kshrc` script executes when you log in and each time you create a subshell.

Because the Korn Shell uses the `.profile` script only for login shells, commands that apply to all shells must be in `$ENV`. The following script sets local shell variables, command aliases, file access permission for files you create, and command history variables. The script is short to help it execute quickly.

```
# Set shell variables
set -o trackall

# Set user file creation mask:
umask 022

# Set command aliases
alias rm='rm -i '
alias rename='mv '

# Set history variables:
HISTFILE=$HOME/.history
HISTSIZE=30
```

Task	Explanation
Set shell variables	The `set` command sets options on how the shell processes commands. The `-o trackall` option tells the Korn Shell to track all commands that you execute. When commands are tracked, the shell remembers where the command is located and finds the command more quickly the next time it is used.
Set user file creation mask	The `umask` command determines the level of security assigned to files you create. In this example, the value `022` lets you create, edit and read files in your directories; other users can only read your files.
Set command aliases	The `alias` command is used to abbreviate long command lines, create new commands, or cause standard commands to perform differently. The first alias enforces the use of the `-i` option with the `rm` command. This option causes the `rm` command to ask whether to delete each file specified in the command line. The second alias creates a new command, `rename`, using the `mv` command. This alias provides DOS users with a more familiar command name for moving files.
Set history variables	The `HISTFILE` variable sets the file name used to store the command history. If this variable is not set, the default file name is `$HOME/.sh_history`. The `HISTSIZE` variable sets the number of previously-executed commands accessible by the shell. The default value is 128 command lines.

For More Information ...

For more information on shell variables, refer to *sh*(1), *ksh*(1), *keysh*(1), and *csh*(1) in the *HP-UX Reference* and the shell tutorials in the *Shells and Miscellaneous Tools* volume of *HP-UX Concepts and Tutorials*. For more information on specific commands used in these example scripts, refer to the *HP-UX Reference*.

Chapter Command Summary

Table 10-1. (For script examples, see the previous sections)

To Do This ...	Type This ...
Edit system login script (Root permission needed)	`vi` /etc/profile
Copy and edit default login script	`cp` /etc/d.profile $HOME/.profile; vi $HOME/.profile
Make sure variables are used in subshells	Include in `.profile`: `export` *VARNAME*

11

Networking with HP-UX

Your HP-UX system can use a variety of networking services to enable you to transfer copies of files to other computer systems. These services can also enable you to log onto remote machines on the network and run commands and processes remotely.

Specifically, this chapter gives you information on using three of the ARPA/Berkeley Services to do the following tasks:

- Copying files to and from a remote computer: `ftp`.
- Copying files remotely: `rcp`.
- Logging onto another computer on the network: `rlogin`.

Copying Files Using ftp

The `ftp` file transfer program allows you to copy files between your local system and remote systems and among remote HP-UX, UNIX, and non-UNIX network hosts that support ARPA services. This service not only allows you to perform remote file copying , but also facilitates file management operations such as changing, listing, creating, and deleting directories on a remote system for which you have a valid login or account.

Using `ftp` you can copy a local file to a remote file or vice versa. You can also append a local file to the end of a remote file. The file to which you are copying can have either the same or a different directory path and/or name as the one on the originating system.

Preparing to Use ftp

- Make sure that your `/etc/hosts` file contains entries for the remote hosts with which you will communicate.
- Have the system administrators for the remote hosts arrange to give you a password and an account, or a login to someone else's account, so that you can log in on the remote hosts.

Transferring Files with ftp

1. To invoke `ftp` and connect to a remote host in one step, type the following:

   `ftp` *remote_hostname*

   This connects you to the remote host. `ftp` then confirms the connection and prompts you for a remote login name:

   `Name (`*remote_hostname*`):`

   If you intend to log in with the same remote login name as you local login name, just press Return.

2. Enter the password associated with your remote login name and `ftp` will confirm this action with a message and a confirmation that you are logged in:

`Password (`*remote_hostname*`):`

`Password required for` *remote_login_name*
`User` *remote_login_name* `logged in.`

3. If you are going to transfer binary (as opposed to "readable" text) files, type `bin` at the prompt, before proceeding.

- Use `get` to transfer files from a *remote host* to your *local directory*.
  - At the `ftp>` prompt, type:

    `get` *remote_file*

    The *remote_file* can be the name of a file in the remote working directory. In that case, `ftp`, copies the file to the local working directory and gives it the same file name. If the file is in another directory on the remote host, *remote_file* is the absolute or relative path for that file. In that case, `ftp` copies the file to a file name with the same path on your local system. If there is no matching path, `ftp` gives you a message, "`No such file or directory`". If the destination file already exists, `ftp` overwrites its contents with the contents of the remote file.

    When copying successfully, `ftp` gives you messages confirming the copy and the length of time it required.

- Use `put` to transfer files from your *local directory* to a *remote host*.
  - At the `ftp>` prompt, type:

    `put` *local_file remote_file*

    - In this case, *local_file* is copied to the remote file name in the specified remote directory.
      - *local_file* can be the name of the local file in your current local working directory. `ftp` will copy the file into a file of the same name in *remote_file*.
      - *remote_file* can be an absolute or relative path to a file name on the remote host. If not specified otherwise, it will be in the current working directory on the remote host.

Note If you have an `ftp` connection to a remote system, you cannot `put` or `get` a file back onto the same system. If you attempt this inadvertently, `ftp` gives you a message, "`No such file or directory.`"

General File-Manipulation Commands for ftp:

Some of the set of `ftp` file-manipulation commands, such as `cd`, `mkdir`, `pwd`, and `rmdir` function in the same way as the corresponding HP-UX commands. Others, such as `append`, `delete`, and `lcd` have functions which are unique to `ftp`.

Table 11-1. File-Manipulation Commands for `ftp`

To Do This ...	Type This ...
Display the name of the current remote working directory	`pwd`
Display the name of the current *local* working directory	`!pwd`
Invoke a shell on the local host	`!`
Copy a local file onto the end of a remote file	`append`
Change the working directory on the remote host to *remote_directory*	`cd` *remote_directory*
Change the working directory on the local host to *local_directory*	`lcd` *local_directory*
List the contents of the current remote directory	`ls`
Create a remote directory	`mkdir` *remote_directory*
Delete a remote directory	`rmdir` *remote_directory*

If you need information on any of the `ftp` commands, just type `help` (or `?`) at the `ftp` prompt.

Exiting ftp

To close the connection with the remote host and exit `ftp`, type:

```
bye
```

Copying Files Remotely Using rcp

You can copy files between HP-UX or other UNIX hosts on the network using `rcp`. Also, using appropriate options, you can copy directories between systems using `rcp`, if the configuration files that the service uses are set up properly.

Preparing to Use rcp

Using `rcp` allows you to copy files and directories to and from a remote host and to copy among remote systems as well. To use `rcp`, you'll need the following prerequisites:

- An account (login) on the remote host.
- A `.rhosts` file in the remote host home directory containing the names of your local host system and your local login name.
- A `.rhosts` file on your local system, as well. This contains the names of all the systems you will copy from. It will ensure that you will be able to use `rcp` when you use `rlogin` on the remote system.
- A `/etc/host` file on your local system which lists hosts with which you can communicate using ARPA/Berkeley Services. For each host, the file has a line containing information about the remote host in the following form:

  internet_address official_name alias

You will find that the `/etc/hosts` file is useful for looking up names and addresses on the network. To facilitate such a lookup, use the `grep` tool described in Chapter 8 in this manual.

For example, an entry in the `.rhosts` file on the remote system might be:

```
hpabc leslie
```

where `hpabc` is the name of your local system and `leslie` is your local login name.

Note It is important to protect your remote `.rhosts` file and home directory to prevent unauthorized users from gaining `rcp` access

to your remote account. Only you should be able to write to the `.rhosts` file.

- Make sure it is owned by you.
- Use `chmod` to set the permission of `.rhosts` to 400 (`-r--------`). See Chapter 12 for details.
- Do the same to protect the rest of your remote home directory with at least 711 (`-rwx--x--x`) permission.

Copying a Local File to a Remote Host

To copy from your system to a remote system, use the following syntax:

`rcp` *local_file remote_hostname*:*remote_file*

Note that, if *local_file* is not in your current directory, you will need to supply the relative path (to get from your current directory) or the absolute path (from /), in addition to the local file name. You will need to specify the complete (absolute) path for the *remote_file* on *remote_hostname* only if you want it to go into a directory other than the remote home directory.

For example, to copy `myfile` from your current directory to a remote system called `xyz`:

```
rcp myfile xyz:/users/leslie/otherdir
```

In this case, `myfile` will be copied as `myfile` into the remote subdirectory, `otherdir`. If you had only supplied the remote host name, `rcp` would have copied `myfile` into the remote home directory, also as `myfile`.

You can also include a filename in the destination. For example, to copy to a system named `xyz`:

```
rcp myfile xyz:/users/leslie/otherfile
```

In this case, you have copied `myfile` as `otherfile`, in the remote directory `leslie`.

Copying a File on a Remote Host to Your Local Directory

Now, to reverse the process, here is how you would copy a file *from* a remote host into your local directory. Use the following syntax:

```
rcp remote_hostname:remote_file local_file
```

For example, to copy `myfile` from your account in a remote system `xyz` into your current directory:

```
rcp xyz:/users/leslie/myfile .
```

The dot (`.`), you'll remember from Chapter 3, is shorthand for "current directory". In this case, `myfile` will be copied as `myfile` from the remote directory into your current directory. You do not have to supply the destination filename if you don't want to copy it to a new name.

If you want to copy `myfile` into another directory in your home system, use a path name, absolute or relative, as shown:

```
rcp xyz:/users/leslie/myfile otherdir/
```

Or, if you want to copy the file to another file name in another directory:

```
rcp xyz:/users/leslie/myfile otherdir/otherfile
```

Run the `ls` command to confirm what you have done.

Copying a Local Directory and its Contents to a Remote System

To copy a local directory with all its files and subdirectories to a remote host, use `rcp` with the `-r` (recursive) option using the following syntax:

```
rcp -r local_dir remote_hostname:remote_dir
```

As before, if *local_file* is not in your current directory, you will need to supply the relative path (to get from your current directory) or the absolute path (from `/`), in addition to the local file name. Also, the *remote_file* will require an absolute path (from `/`).

For example, you may want to copy a directory of miscellaneous files called `work` to a remote computer host called `abc`. To copy the entire `work` directory to a directory which is already there called `products`, type the following:

```
rcp -r work abc:/users/leslie/products
```

You have created a directory named `work`, with all its contents, in `abc:/users/leslie/products`.

Again, this assumes that you are doing this while you are in the local directory which contains `work`. Otherwise, you would have to give a relative or absolute path to that directory, along with its name, such as `/users/leslie/work`.

Copying a Remote Directory and its Contents to a Local System

To copy a remote directory with all its files and subdirectories to a local directory, use `rcp` with the `-r` (recursive) option in the following syntax:

```
rcp -r remote_hostname:remote_dir local_dir
```

For example, you may want to copy a remote directory called `work` to your current directory. To copy the entire `work` directory, type the following:

```
rcp -r abc:/users/leslie/work .
```

The dot (`.`) signifies the current directory, where the `work` directory will be copied. Run the `lsf` command to confirm that this has been done.

Logging in on Another Computer Using rlogin

If you have an account on a **remote host**, then you can use `rlogin` to log in on a remote host by supplying your remote login name and password. You can then work on that system just as you would on your home system.

If the remote host is configured to allow it, you can also log in on a remote host automatically, without having to supply your login name and password.

Logging in on a Remote Host

- At the shell prompt, use the form:

  `rlogin` *remote_hostname*

  The *remote_hostname* is the name of an appropriately configured remote system. As before, this system is named in your `/etc/hosts` file and in your `.rhosts` file. The remote host prompts you for your remote password.

- Enter your remote password. The remote host logs you in with the login message and the remote host prompt.

If for some reason you should make an error in entering your password, the remote host will give you the error message, `Login incorrect`, and will prompt you for your login, and your password:

```
Login incorrect
login:
```

Getting the Same Working Environment on the Remote Host

To get the remote host environment to behave in the same way as you are used to having your home environment work, you can set the `.profile` or `.login` values the same by copying your local `.profile` or `.login` files to your home directory on the remote system. As with your home system, the values in your `.profile` or `.login` will take precedence over the values in the remote system's `/etc/profile` or `/etc/csh.login` file.

Logging Out and Exiting the Remote Host

You can log out of the remote host just as you would from your home system, by typing:

```
exit
```

Typing CTRL-d also logs you out on most system.

At this point you are logged out of the remote host, disconnected, and returned to HP-UX on your local system, which displays a message and your local prompt:

```
Connection closed.
$
```

Temporarily Returning to Your Local System

If you wish to execute a command on your local command while you are in `rlogin`, type the `rlogin` escape character (normally a ~) followed by ! and the command. (The ~ will be invisible until you type the ! after it.) After the command has executed, `rlogin` returns you to the remote host. For example:

```
~! pwd
/users/leslie
[Returning to remote]
```

Press Return, or enter a command to redisplay the remote host prompt.

For More Information ...

Refer to the HP books, *Using Network Services* and *Using ARPA Services*.

Chapter Command Summary

Table 11-2. Commands

To Do This ...	Type This ...
Invoke `ftp` and connect to *remote_host*	`ftp` *remote_host*
Copy files from *remote_host* to current directory, in `ftp`	`get` *remote_file*
Copy files from your local current directory to the current directory on *remote_host* in `ftp`	`put` *local_file*
List the contents of the current remote directory	`ls`
Exit `ftp`	`bye`
Also see Table 11-1 for File-Manipulation Commands on `ftp`	
Copy a file from your *local_file* to a file on a remote host, using `rcp`	rcp *local_file remote_hostname:remote_file*
Copy a file from a remote host to your local directory, using `rcp`	rcp *remote_hostname:remote_file local_file*
Copy a directory structure from your local system to a remote host	rcp -r *local_dir remote_hostname:remote_dir*
Copy a directory structure from a *remote* system to your local system	rcp -r *remote_hostname:remote_dir local_dir*
Log in on a remote system	`rlogin` *remote_hostname*
Exit `rlogin`	`exit`

12

Making Your System Secure

HP-UX provides many security features to protect files from unauthorized access. However, you need to follow good security practices to maintain security on your system. The degree to which you need to enforce security measures depends on where you work, the security policy in your workplace, and the type of information with which you work.

Security Strategies

This chapter summarizes the security strategies you should follow to help keep your system secure:

- Become familiar with the security policies of your workplace.
- Keep your terminal secure.
- Choose a secure password, and protect your password after you've chosen it.
- Be aware of who has permission to access your files and directories, and be able to control such access.

Note Security requires ongoing attention, and it may be impossible to have a 100% secure system under all circumstances. Therefore, this chapter gives you some guidelines for securing your system. However, even these cannot guarantee you a completely secure system.

Securing Your Terminal

12

When you are working with sensitive material, take care to position your terminal so the screen is not visible to others. Never leave your terminal unattended. Log off (`exit`) when you leave your terminal, or, if you're using VUE or the X Window System, lock your screen.

Guidelines for Securing Your Terminal

When working with sensitive material, take these security precautions:

- Position your terminal so the screen points away from open windows and doors.
- Never leave your terminal in a non-secure state:
  - Exercise care when logging in. Make sure no unauthorized person is observing you while you're entering your password.
  - Log off if you will be away from your terminal for a time.
  - Clear your display, even if you leave your terminal for a brief period. Type `clear` at the command line prompt. (Note that the `clear` command clears only the current screen; one can still scroll up and see previous screens.)

Note Check the security policies in your workplace. You may be required to log off whenever you leave your terminal, even if only for a brief period.

Working in an Audited Environment

HP-UX provides the capability to audit computer use, both on an individual and system-wide basis. Depending on how your system is configured, your actions may be recorded by an audit program. This subsystem monitors user actions at your terminal and records security-relevant information.

Choosing a Secure Password

12

When you choose a password, you want to ensure that no one can guess what you chose. If someone knows your password, that person may log in and access your files. This section offers suggestions on how to select and protect your password. These guidelines are of particular significance if you work with sensitive material.

Selecting a Secure Password

When selecting a password in a secure environment, follow these guidelines:

- Choose a password that isn't publicly associated with you (your personal or professional life, your hobbies, etc.):
  - Don't use your name, your spouse's name, your children's names, or your pets' names.
  - Don't use the name of your street or your car.
  - Don't use phone numbers or special dates (anniversaries, birthdays, etc.).
  - Don't use your address, social security number, or license plate numbers.
- Choose a password that isn't listed in the dictionary (spelled either forwards or backwards). Password-cracking programs can use dictionary lists.

What *can* you use as a password? Here are a few suggestions:

- Make up a nonsense word.
- Make up an acronym.
- Misspell a word intentionally.
- String together syllables from a favorite song or poem.

Note HP-UX requires that your password be six to eight characters long. At least two of these characters must be letters (uppercase or lowercase); at least one character must be either a numeral (the digits 0 through 9) or a special character (such as -, _, or $). See Chapter 1 for some examples.

Protecting Your Password

When you have chosen your password, follow these guidelines to ensure that no one discovers it:

- Never write down your password.
- Don't tell others your password.
- Don't let others watch as you type your password.
- Don't store your password in the function keys of a terminal.
- Change your password occasionally (for example, once every three or four months). Refer to "Setting Your Password" in Chapter 1, if you need information on how to change your password.
- If you use more than one computer, use a different password for each system.

Protecting Your Files and Directories

12

Access permissions determine who can access your files and directories and the type of access allowed. You should always be aware of the permissions assigned. Check your files and directory permissions periodically to make sure appropriate permissions are assigned. If you find any unfamiliar files in your directories, report them to the system administrator or security officer.

Always carefully consider the permissions you allow on your files and directories. Give others access to them *only* when you have good reason to do so (if you are working on a group project, for example, your group may need access to certain files or directories.)

As you learned in Chapters 3 and 4, the basic access permissions assigned to files and directories distinguish between three classes of users: owner, group, and other.

Each of the classes of users can access files or directories in any of three ways: read, write, and execute/search (`r`, `w`, and `x`). You can list the access permissions of your files and directories by using the `ll` command.

For information on access control lists (ACLs), which allow finer control of access to files, see *acl*(5) in *HP-UX Reference* and *HP-UX System Security*.

Access to Sensitive Files

Make sure that permissions assigned to sensitive files and directories are appropriate. Here are some general suggestions:

- Only you should be able to write to your home directory.
- Only you should be able to write to the files used to customize your home environment, for example, `.login` and `.profile`. (`.profile` is discussed in Chapter 10.)
- Only you (and the pseudo-group "mail," assigned to the mailer) should be able to write to your mailfile `/usr/mail/`*username*.

For More Information ...

For details on determining permissions, by using the `ll` command, see the section in Chapter 2 "Finding Out Who Can Use Your Files" or the section in Chapter 3 "Displaying Directory Permissions: The `ll` Command," in this manual.

Changing Who Has Access to Files

If you want to change the basic access permissions assigned to a file use the `chmod` (change `mode`) command to control who has read, write and execute permission to your files.

12

You can change file permissions using the `chmod` command. The `chmod` command sets a file's read, write, and execute permission for you, the file's group, and other users. Before using this command, you should always carefully consider what file permissions you give to others.

In general, give others access to your files *only* when you have good reason to do so (if you are working in a group project, for example, your group may need access to certain files).

Using chmod to Set File Permissions Numerically

To illustrate the `chmod` command, you can set the permissions for `myfile` so that *only you* can read from and write to the file. A commonly-used syntax for `chmod` is as follows:

`chmod` *number file_name*

Number is a three-digit number specifying what permissions you want to assign to the file. Each of the three digits sequentially sets permissions for each of the three classes: the owner, the group to which the owner belongs, and all other users. The *file_name* is the name of the file you want to protect.

Also, each of the three digits can be thought of as made up of a sum of one or more of the following, taken separately for each of the three digits:

```
0   no permission (-)
1   execute (x) permission only
2   write (w) permission only
4   read (r) permission only
```

Then, by this scheme:

- 400 gives "read" (`r--------`) permission for you, the user, only.
- 440 gives "read" (`r--r-----`) permission for you and your group.

- 700 (4 + 2 + 1, 0, 0) gives "read, write, and execute" (`rwx------`) permission for you, but no permissions to anyone else.

Using chmod to Set File Permissions Symbolically

You can also specify permissions for `chmod` using the letters `u`, `g`, and `o`, as symbolic code for the owner ("user"), group, and others (the *class*). This "symbolic mode" is an easier scheme to remember than the numeric mode, since the symbols `r`, `w`, and `x` (the *mode*) are used directly as arguments in the command. Setting up the `chmod` syntax is a little different, making use of the `+`, `-`, and `=` signs. The syntax is:

`chmod` *class* [`+` | `-` | `=`] *mode*, [...] *file_name*

For example, you can use the symbolic mode to create 644 (`rw-r--r--`) permissions by specifying the symbols `rw`, `r`, and `r` directly in the `chmod` command. "User" is represented by `u`, "group" by `g`, and "other" by `o`. To assign the permissions *absolutely*, use the `=` sign in the argument. Unspaced commas separate class-permissions:

```
chmod u+rw,g+r,o+r file_name
```

To create 600 (`rw-------`) permissions and set "no permissions" for the classes `g` and `o`., use `=` with no symbol following:

```
chmod u=rw,g=,o= file_name
```

Permissions are *added* with the `+` sign. Again, separate each class-permission with a comma and no space:

```
chmod u+rw,g+r,o+r file_name
```

You can also *subtract* permissions from `u`, `g`, or `o`, using `-`, if you just want to restrict the level of permission from a previous "higher" level. For example, if you had set `rwxrw-rw-` and you wanted to change this to `rwx------` (`g` represents "group" and `o` represents "other"):

```
chmod g-rw,o-rw file_name
```

However, unless you began with *no* permissions you may find that using `+` or `-` has added to, or subtracted from, some previously existing permissions for

that file. Run the `ll` command to check this. If in doubt, set the permissions absolutely by using =.

For example, suppose you want to protect `myfile` so that neither you nor anyone else can modify it, but everyone can still read from it:

```
$  chmod 444 myfile
```

or

```
$ chmod u=r,g=r,o=r myfile
```


When permissions are being set the same, you can also combine the arguments as:

```
$  chmod ugo=r myfile
```

With only read permission on `myfile`, no one can write to it. Also, if you now try to remove `myfile`, the `rm` command asks you whether you really want to remove the file:

```
$ rm myfile
myfile: 444 mode? (y/n) n
```

If you do not *want to remove it, enter* `n`.
If you do *want to remove it, enter* `y`.

Later, if you want to permit yourself and members of your group to read from and write to `myfile`, use `chmod` as follows:

```
$ chmod 664 myfile
```

or

```
$ chmod ug=rw,o=r myfile
```

The `ll` command now should show:

```
-rw-rw-r--  1  leslie  users      154 Nov  4 10:18 myfile
```

For More Information ...

This section covered some of the most common uses of the `chmod` command for protecting files. To learn more about `chmod`, refer to the following:

- The manual *System Administration Tasks*
- The *chmod*(1) entry in the *HP-UX Reference*.

Table 12-1 summarizes the various chmod commands you can use to protect myfile.

Table 12-1. Summary of Uses of chmod

12

To Set Permissions so that ...	Type This ...	
	Numeric:	Symbolic:
Only you can read from myfile, and no one (including you) can write to it. Set permissions to -r--------.	`$ chmod 400 myfile`	`$ chmod u=r,g=,o= myfile`
Everyone can read from myfile, but no one can write to it. Set permissions to -r--r--r--.	`$ chmod 444 myfile`	`$ chmod ugo=r myfile`
Only you can write to myfile, but everyone can read it. Set permissions to -rw-r--r--.	`$ chmod 644 myfile`	`$ chmod u=rw,go=r myfile`
Only you and members of your group can write to myfile, but everyone can read it. Set permissions to -rw-rw-r--	`$ chmod 664 myfile`	`$ chmod ug=rw,o=r myfile`
Everyone can read from or write to myfile. Set permissions to -rw-rw-rw-.	`$ chmod 666 myfile`	`$ chmod ugo=rw myfile`
Only you can read from or write to myfile, but no one else can. Set permissions to -rw-------	`$ chmod 600 myfile`	`$ chmod u=rw,go= myfile`

Changing Who Has Access to Directories

In addition to changing permissions on files, the `chmod` command can also change permissions on directories. Using `chmod`, you can control who has access to your directories, and what kind of access they have. For example, you can protect a directory so that no one can list its files. Or you can control whether users can remove, rename, or alter files in a particular directory.

Table 12-2 defines some of the more common uses of `chmod` with directories. All the examples in this table assume that the directory `projects` exists under your current working directory.

Table 12-2. Setting Directory Protection for the projects Directory

To Set Permissions to ...	Type This ...	
	Numeric:	**Symbolic:**
Allow other users to list and access the files in `projects`, but not to create or remove files from it. Set permissions to `drwxr-xr-x`.	`$ chmod 755 projects`	`$ chmod u=rwx,go=rx projects`
Allow all users to list, create, remove, and access files in `projects`. Set permissions to `drwxrwxrwx`.	`$ chmod 777 projects`	`$ chmod ugo=rwx projects`
Allow only yourself to list, create, remove, and access files in `projects`. Set permissions to `drwx------`.	`$ chmod 700 projects`	`$ chmod u=rwx,go=- projects`

Note

When determining who should be allowed to use your directories, be aware that *anyone who can write to a directory also can remove or rename a file in that directory*—even if that person cannot write to the file.

Controlling Default Access Permissions

In the preceding section you learned how to change the permissions on individual files and directories using the `chmod` command. You should also be aware of the default permissions assigned to all of your files and directories at the time you create them. You can list or change the default permission settings by using the `umask` command.

Default file permissions are assigned by the system whenever you create a new file or directory, and these are governed by your umask setting. Unless set up otherwise by you or your system administrator, your default `umask` setting will be 0, which means that new *files* you create will have read/write permission for everyone (666 or -rw-rw-rw-) and new *directories* you create will have read/write/search permission for everyone (777 or `drwxrwxrwx`).

You may want to change your `umask` setting to a non-zero value, to make the default access permissions to your newly-created files and directories more restrictive. You can do so using the `umask` command and either the numeric or, in the Korn Shell, symbolic arguments. For simplicity, the following shows you how to use `umask` with numeric arguments.

The number given as a parameter to the umask command works in an opposite manner to the number given to the `chmod` command. The "mask" serves to remove permissions as opposed to granting them. That is, the digits in the `umask` number are "subtracted" from 666 for files and 777 for directories when you are creating their initial permissions. For example, suppose you enter:

```
$umask 022
```

Now when you create new files their default permissions will be 644 (`-rw-r--r--`). When you create new directories their default permissions will be 755 (`drwxr-xr-x`). If the `umask` value were instead set to 077, your default file permissions would be 600 (`-rw-------`) and your default directory permissions would be 700 (`drwx------`).

To find out what `umask` you currently have, type:

```
umask
```

Note If you are using the Korn Shell or Key Shell, you can also enter the `umask` permissions symbolically. In the case above, you can type: `umask u=rwx,g=rx,o=rx`, with results as shown above for files and directory permissions. Check which shell-type you have by typing `echo $SHELL`, from you login shell, before attempting this.

Here are some examples of common settings for umask:

`umask 077` Assigns permissions so that only you have read/write access for files (read/write/search for directories) you own. All others have no access permission to your files and directories.

`umask 022` Assigns permissions so that only you have read/write access to files (read/write/search access to directories) you own. All others have read access only to your files (read/search access to your directories).

`umask 002` Assigns permissions so that only you and members of your group have read/write access to files (read/write/search access to directories) you own. All others have read access only to your files (read/search access to your directories).

Caution You should not set a `umask` value, such as 2xx or 7xx, which restricts your access permissions to your own files. A number of HP-UX utilities, such as `vi`, assume that you can always access newly-created files. Such files might include the temporary files which `vi` creates. These utilities may malfunction when used under such a restrictive `umask` setting.

If you set `umask` at a shell prompt, it will apply to shells and subshells in the current login session only. It won't apply to future login sessions. To apply a `umask` setting automatically at login, add the `umask` command to your `.profile` (Korn and Bourne Shell users) or `.login` file (C Shell users).

For More Information ...

To get more information about the `umask` command refer to the *umask*(1) entry in the *HP-UX Reference*).

To learn more about the `.profile` and `.login` files see Chapter 10 in this manual, and *Shells: User's Guide*.

Chapter Command Summary

Table 12-3. Commands

To Do This ...	Type This ...
Display file permissions	`ll`
Numerically change file or directory permissions	`chmod` *number name*
Symbolically add or subtract file or directory permissions	`chmod` *class*±*permissions name*
Symbolically change (absolutely) file or directory permissions	`chmod` *class*=*permissions name*
Find out permissions mask setting	`umask`
Numerically change permissions mask setting	`umask` *mask_number*

A

Using the C Shell Environment

Use this section for reference to the C Shell.

- The default system login script for the C Shell—`/etc/csh.login`.
- The default user login script for the C Shell—`/etc/d.login`.
- An example C Shell `.login` script.
- The default user run-command script for the C Shell—`/etc/d.cshrc`.
- An example C Shell `.cshrc` script.
- Using command history in the C Shell

Logging in with C Shell

If you use the C Shell, remember that it has a system login script and two user login scripts:

- The `/etc/csh.login` is your system login script.
- The `.cshrc` script executes after the system login script and each time you create a subshell.
- The `.login` script executes after the `.cshrc` script, but only during login.

The Default System Login Script for the C Shell: /etc/csh.login

```
# @(#) $Revision: 56.2 $
# Default (example of) system-wide profile file (/bin/csh initialization).
# This should be kept to the bare minimum every user needs.

     # default path for all users.
     set path=(/bin /usr/bin /usr/contrib/bin /usr/local/bin)
     set prompt="[\!] % "
     setenv TZ MST7MDT                          # change this for local time.

     if ( ! $?TERM ) then                       # if TERM is not set,
             setenv TERM hp                     #   use the default
     endif

# This is to meet legal requirements...
     cat /etc/copyright                         # copyright message.
# Miscellaneous shell-only actions:
     cat /etc/motd                              # message of the day.

     if ( -f /bin/mail ) then
             mail -e                            # notify if mail.
             if ( $status == 0 ) echo  "You have mail."
     endif

     if ( -f /usr/bin/news ) then
             news -n                            # notify if new news.
     endif

     if ( -r /tmp/changetape ) then             # might wish to delete this:
             echo
             echo "You are the first to log in since backup:"
             echo "Please change the backup tape.\n"

             rm -f /tmp/changetape
     endif
```

Login Script Task	Explanation
Set `path`	The default `path` tells your shell to search `/bin`, `/usr/bin`, `/usr/contrib/bin`, and `/usr/local/bin`. The current directory is not included. The `set` command sets `path` as a local shell variable. The C Shell has an automatic mechanism that sets the environment variable `PATH` equal to the shell variable `path`.
Set `prompt`	The shell variable `prompt` is a local shell variable in the C Shell. This script sets `prompt` to "`[`*n*`] % `" where *n* is the command event number. Command event numbers are used with the C Shell's command history feature.
Set time zone (`TZ`)	The `TZ` variable is set to Mountain Standard Time. If you are in another time zone, your `/etc/csh.login` script should be different from this example.
Set `TERM`	If the `TERM` variable is not set, this script sets it to `hp`.
Display `copyright` file	The `cat` command displays the contents of `/etc/copyright`. This file contains copyright messages required by law.
Display `motd` file	The `cat` command displays the contents of `/etc/motd`, if it exists. This file contains the *m*essage *o*f *t*he *d*ay.
Check for mail	If the file `/bin/mail` exists, the `mail -e` command checks whether you have mail. If you have mail, the "`You have mail`" message is displayed.
List news items	If the file `/usr/bin/news` exists, the `news -n` command is executed. This option displays the list of news files in `/usr/news`, rather than displaying the contents of the files.
Check for backup tape	If the file `/tmp/changetape` exists and is readable, the message, "`Please change the backup tape`" is displayed. Check your site's backup procedures before changing the tape.

The Default Local Login Script for the C Shell: /etc/d.login

The C Shell uses a user login script named `.login` located in the home directory. The default user login script is based on the file `/etc/d.login`.

The C Shell user login script is similar to the Bourne Shell/Korn Shell script. The default user login script resets the value of `PATH`, sets terminal characteristics, and sets shell variables.

A

```
# @(#) $Revision: 64.2 $

# Default user .login file ( /bin/csh initialization )

# Set up the default search paths:
   set path=(/bin /usr/bin /usr/contrib/bin /usr/local/bin .)

# Set up the terminal type:
   eval 'tset -s -Q -m ':?hp' '
stty erase "^H" kill "^U" intr "^C" eof "^D" susp "^Z"
hupcl ixon ixoff tostop
tabs

# Set up shell environment:
   set noclobber
   set history=20
```

Default Script Task	Explanation
Set path	The value of `PATH` is explicitly changed, meaning the entire value of the variable is reset. The new value adds the current directory to the end of `PATH`. If you attempt to execute a command that HP-UX cannot locate along any of the other paths, this value of `PATH` causes the shell to search for the command in the current directory (.).
Set terminal characteristics	The `tset` command generates the `TERM = (hp)` prompt during login. Based on the value you enter, the `TERM` environment variable is set and the terminal's characteristics are defined. If you press Return at the prompt, `TERM` is set to `hp`. The `stty` command sets options for key definitions, modem connections, and tab settings. Refer to *stty*(1) for more information.
Set shell variables	The local shell variables `noclobber` and `history` are set. The shell variable `noclobber` restricts output redirection, so the contents of files cannot be destroyed accidentally. The shell variable `history` creates your command history buffer and sets its size. In this script, `history` is set to record the last 20 commands executed. If this variable is not set, you have no command history and cannot edit and reuse previously-executed commands.

Some Differences in Using the C Shell

The `/etc/d.login` differs slightly in syntax and function from the `/etc/d.profile` script used by the Korn and Bourne Shells. The more obvious differences include the following:

- The C Shell does not have the `set -u` command.
- You don't need a logout trap in the C Shell because the C Shell automatically echoes "`logout`" and executes a logout script (if it exists) when you exit your login shell.
- The `EDITOR` variable is not set, but the `noclobber` and `history` variables (which are C Shell variables) are set.

An Example C Shell .login Script

The `.login` script is run after the `.cshrc` script when you log in.

Like the Korn Shell, the C Shell distinguishes between environment variables and shell variables. Environment variables, defined in the `.login` script, are used by your login shell and are passed to subshells.

To copy the default script into your home directory, type these commands:

```
% cd                          Changes to your home directory.
% cp /etc/d.login .login      Copies the default script.
%
```

A

You can now edit this file for your own needs. The following script sets terminal characteristics, the global environment variable `PATH`, and the local shell variable `prompt`. The script also displays a login message.

```
# Set up the terminal type:
eval 'tset -s -Q -m 'console:300h' -m '=1200:pc''

# Set PATH:
set path=(/bin /usr/bin /usr/local/lib /usr/lib . $HOME/mybin)

# Set local shell variables:
set prompt="[\!] "

# Provide login message:
echo 'grep $LOGNAME /etc/passwd | cut -d: -f5' -- have a nice day.
```

Example Script Task	Explanation
Set terminal characteristics	The `tset` command sets up your terminal when you first log in to an HP-UX system. With the `tset` command, the login script can select between two or more terminal types. This example chooses between a high-resolution graphics display station (`300h`) that functions as the console or a personal computer (`pc`) logging in via a 1200 baud modem. To modify this command for your own use, replace `console` and `=1200` with the descriptions you need. For example, you might set one option to `=9600` (equals 9600 baud) and the other to `<9600` (less than 9600 baud). Then replace the `300h` and `pc` with the terminal types you use. (Refer to *tset*(1) for more information).
Set `PATH`	The `set` command sets local environment variables. `PATH` is a global variable, and `path` is a local variable in the C Shell. The C Shell automatically sets `PATH` equal to `path`. This example explicitly sets the entire value of `path`. The dot (`.`) represents the current directory. `$HOME/mybin` has been added to the default paths.
Set local shell variables	The `set prompt` command redefines the default prompt (`%`) to show the event number being typed. This feature is useful with the command history mechanism. As set here with `[\!]`, the prompt will consist of an event number *n*.
Provide login messages	The `echo` commands write messages onto the terminal screen. This message echoes the user's name and the message "have a nice day." The user's name is extracted from the `/etc/passwd` file. The `grep` command finds the entry that contains `$LOGNAME`. The `cut` command extracts the fifth field which should be the user's name.

The Default Local Run Command Script for the C Shell: /etc/d.cshrc

In the C Shell, the user run-command script is `.cshrc`. (The `rc` in the file name refers to the term *r*un *c*ommand.) When you first log in, the C Shell executes the `.cshrc` command in your home directory before it executes your `.login` script. Additionally, the `.cshrc` script runs each time you create a subshell.

The default version of this script is `/etc/d.cshrc`. This script sets a restricted version of path for your subshells, sets local shell variables, and defines command aliases.

A

```
# Default user .cshrc file (/bin/csh initialization).
# Usage:  Copy this file to a user's home directory and edit it to
# customize it to taste.  It is run by csh each time it starts up.
# Set up default command search path:
#
# (For security, this default is a minimal set.)
     set path=( /bin /usr/bin )

# Set up C shell environment:
     if ( $?prompt ) then                  # shell is interactive.
         set history=20                    # previous commands to remember.
         set savehist=20                   # number to save across sessions.
         set system=`hostname`             # name of this system.
         set prompt = "$system \!: " # command prompt.

         # Sample alias:
         alias       h       history

         # More sample aliases, commented out by default:
            # alias     d       dirs
            # alias     pd      pushd
            # alias     pd2     pushd +2
            # alias     po      popd
            # alias     m       more
     endif
```

Task	Explanation
Set PATH	A restricted `path` is defined that includes only `/bin` and `/usr/bin`. Since `.cshrc` executes before `.login` during the login process, this restricted value of `path` is overridden by the path defined in `$HOME/.login`. When subshells are spawned, the restricted value of `path` is used.
Set shell variables	If the shell is interactive, then various shell variables are set. The shell variable `history` creates your command history buffer and sets its size. In this script, `history` is set to record the last 20 commands executed. If this variable is not set, your subshells have no command history and you cannot edit and reuse previously-executed commands. The shell variable `savehist` tells the shell to record the last 20 commands in the `$HOME/.history` file. This file is used as the starting history for the next session. A shell variable `system` is set to the value output by the `hostname` command. The shell variable `prompt` is set to the value in the `system` variable, followed by the command event number and a colon.
Set command aliases	If the shell is interactive, then command aliases are set. A command alias is an abbreviation for a long command line. This script sets an alias of `h` for the `history` command. The `/etc/d.cshrc` script also contains example aliases that are commented out by default. To use these aliases, simply remove the pound sign (`#`) at the beginning of the line, changing the line from a comment to a command.

To copy the default script into your home directory, type these commands:

```
% cd                          Changes to your home directory.
% cp /etc/d.cshrc .cshrc      Copies the default script.
%
```

An Example C Shell .cshrc Script

If the C Shell is your login shell, it will look for a `.cshrc` script and a `.login` script in your home directory. When you log in, the C Shell runs the `.cshrc` script before it runs the `.login` script. Additionally, the `.cshrc` script is run each time you create a subshell. The following example script sets local shell variables, command aliases, and command history variables.


```
# Set local shell variables:
set noclobber
set ignoreeof
set notify
set autologout 60

# Set command aliases
alias h      'history \!* | more'
alias cd     'cd \!*; dirs'

# Set history variables:
set history=40
set savehist=40
```

Task	Explanation
Set local shell variables	The C Shell offers many predefined shell variables. The following variables are set in the preceding `.cshrc` script example: ■ `noclobber` prevents the contents of a file from being overwritten accidentally. ■ `ignoreeof` forces you to end your session with the `exit` or `logout` command. Using CTRL-D is not allowed. ■ `notify` tells the shell to notify you immediately when background processes are complete. ■ `autologout` automatically logs you out if you do not use the system for the specified number of minutes.
Set command aliases	The `alias` command abbreviates long command lines, creates new commands, or causes standard commands to perform differently. The first alias defines a command that pipes the command history buffer through the `more` command. The second alias replaces the `cd` command with a command that changes directories, then displays the current directory path name. The aliases for `history` and `cd` both use `\!*`. This combination of characters tells the C Shell to substitute any arguments to the command at that point in the alias. For example if you type `cd /users`, `/users` is substituted into the alias where `\!*` is.
Set history variables	The command history buffer saves the commands you type and lets you reuse and edit them. The history file is `$HOME/.history`. To use the history buffer in the C Shell, the `history` variable must be set. Setting the `history` variable to 40 tells the C Shell to record the last 40 commands in the history file. The `savehist` variable tells the C Shell to save the last 40 commands and use them as the starting history for the next session.

A

Setting Environment Variables on the Command Line

In the C Shell, environment variables are assigned new values with the `setenv` command:

`setenv` *name value*

where *name* is the variable name and *value* is the value assigned to the variable.

For example, the following command sets the `PATH` variable to search the `/bin`, `/usr/bin`, and current directories:

A

```
% setenv PATH /bin:/usr/bin:
% _
```

Including : as the last element makes the current directory the last one searched.

In the C Shell, shell variables are assigned with the `set` command:

`set` *name=value*

The `path` variable may be set to the same value in the example above by using the `set` command:

```
% set path=(/bin /usr/bin .)
% _
```

Including . as the last element makes the current directory the last one searched.

Using Command History in the C Shell

To see an example of command history operations in the C Shell, enter the C Shell:

```
$ /bin/csh
% _
```

The cursor changes to indicate the change in shells.

Activate the command history to store 20 commands:

```
% set history=20
% _
```

Now type in some commands:

```
% date
Wed Mar 29 10:57:28 MST 1989
% pwd
/users/terry
% hostname
hpfcma
% _
```

Now type in this command:

```
% history 4
    21 date
    22 pwd
    23 hostname
    24 history 4
% _
```

Notice that the C Shell displays the last three commands (**date**, **pwd**, and **hostname**) *and* the **history** 4 command. You can increase the amount of the command history shown by increasing the number that follows **history**. For example, this will display the last 100 commands if there are 100 commands in the history:

```
% history 100
```

If there are fewer than 100 commands in the history, the full contents of the history will be displayed.

Re-executing the Previous Command in the C Shell

To re-execute the previous command, type !!.

```
% pwd
/users/terry
% !!
pwd
/users/terry
% _
```

A

Recalling and Re-executing Commands in the C Shell

To recall and re-execute an earlier command, type ! followed by the number of the command.

```
% history 4
    25 history 100
    26 pwd
    27 pwd
    28 history 4
% !26
pwd
/users/terry
% _
```

You can also recall a command by typing ! followed by the initial characters of the command string.

```
% !ho
hostname
hpfcma
% _
```

Editing and Re-executing the Most Recent Command in the C Shell

To recall, edit and re-execute the most recent command, use the ^ symbol to delimit the portion of the command to be changed, following it with the corrected text.

```
% ll /dve | grep '^d' | more
/dve not found.
```

```
% ^ve^ev
ll /dev | grep '^d' | more
drwxr-xr-x   2 root     sys     24    Apr 23  10:00 dsk
drwxr-xr-x   2 root     sys           2048  Feb  3  10:02 pty
drwxr-xr-x   2 root     sys           2048  Feb  3  10:02 ptym
drwxr-xr-x   2 root     sys           24    Feb  3  10:00 rdsk
% _
```

Editing and Re-executing an Earlier Command in the C Shell

Recall, edit and re-execute an earlier command by using the :s modifier. In this example, the command wc -l is substituted for more in the original command.

```
% history 3 31 u /dve | grep '^d' | more
    31 ll /dev | grep '^d' | more
    32 pwd
    33 history 3
% !31:s/more/wc -l/
ll /dev | grep '^d' | wc -l
     4
% _
```

Glossary

absolute path name
The name of a file which lists all the directories leading to it, starting with root ("/") and ending with the filename itself.

access permissions
See **file access permissions**.

ACL
Access control list

ANSI
American National Standards Institute

append
A file redirection operation which adds the contents of one file onto the end of another, using the >> symbol. For example `cat file1 >> file2` appends `file1` onto the end of `file2`, so that the new `file2` is a combined file, and `file1` is unchanged.

application
A program used to perform a particular task, usually interactively, such as computer-aided design, text editing, or accounting.

argument
The part of a command line which identifies what (file, directory, etc.) is to be acted upon.

ARPA
Advanced Research Projects Agency, which developed ARPANET in the 60's.

ASCII
American Symbolic Code for Information Interchange.

backup
A copy of all or part of the file system.

back up
To "back up the file system" refers to the process of copying the file system to a separate disk or to a tape.

BASIC
Beginner's All-purpose Symbolic Instruction Code.

bit
BInary digiT

boot
To start up your system, loading it into the computer memory.

boot ROM
A read-only memory which is incorporated into a terminal for the purpose starting the operating system, testing the terminal, and producing a standard display.

byte
A fundamental character-code unit, usually consisting of 8 or 16 bits.

C
A standardized and highly-portable computer language; also the name of the NLS default language/environment (formerly `n-computer`).

character
An element used for the organization, control, or representation of text. Characters include graphic characters and control characters.

CD-ROM
Compact Disc Read-Only Memory.

child process
A new process created by a pre-existing process. The pre-existing process is the **parent process** of the new process.

child directory
See **subdirectory**.

cluster node
A member of a group of workstations connected via a LAN. One computer, the **cluster server**, performs as a server to the cluster.

cluster server
A computer which provides file access, login access, file transfer, printing and other services across a network to a defined cluster of systems (**cluster nodes**) connected via a LAN.

command
A set of instructions you enter into the system to cause it to perform a task.

command interpreter
A program which reads lines of text from standard input (typed at the keyboard or read from a file), and interprets them as requests to execute other programs. An HP-UX command interpreter is called a shell.

command history
The sequence of commands which you have entered, which is stored in .sh_history, and is recalled, line by line, on the command line by pressing ESC twice and using `vi` scrolling commands.

command prompt
A message displayed by the shell indicated that keyboard input is needed. Also called **command line prompt**.

CRT
The system display (formerly called a "cathode ray tube").

current directory
See **current working directory**.

current working directory
The directory in which relative path name searches begin. It is also called the **current directory** or **working directory**.

default search path
The sequence of directory prefixes that *sh(1)*, *time(1)*, and other HP-UX commands apply in searching for a file known by a relative path name (that is, a path name not beginning with a slash (/)). It is defined by the environment variable PATH.

directory
A table of identifiers and references (such as file names) that refer to corresponding files and items of data. Used in a typical HP-UX organizational structure to provide an organizational and logical identity for a given group of files and directories.

DOS
Disk operating system.

environment
The set of defined shell variables (some of which are PATH, TERM, SHELL, EXINIT, HOME) that define the conditions under which your commands run. These conditions can include your terminal characteristics, home directory, and default search paths.

environment variable
A shell variable that determines your terminal's behavior in general, including default search paths, type of prompt, type of editor, and terminal type.

EOF
End-of-file.

file access permissions
A set of permissions which every file in the directory hierarchy has. These permissions are used in determining whether a process can perform a requested operation on the file (such as opening a file for writing). Access permissions can be changed subsequently through the *chmod*(1) command or the *chmod(2)* call.

file
A named storage location in which you can store text or other data.. Files are the basic means which HP-UX uses to organize information.

fileset
Describes a logical, defined set of files on an update or installation tape

file system
The organization of files and directories on a hard disk.

filter
A command that reads data from the standard input, performs a transformation on the data, and writes it to the standard output.

Hex
Hexadecimal

home directory
This is the directory where the user starts after logging in. It is the directory name given by the value of the shell variable HOME.

host name
A string which uniquely identifies a machine in a network. There are generally different **host names** associated with different networks.

HP-UX cluster
A group of workstations connected via a LAN. One computer, the **cluster server**, performs as a server for the **cluster nodes**.

invisible file name
A file name in which the first character is a dot (.) and which is not displayed by `ls` or by `ll` without added options, such as `-a`.

kernel
The part of the HP-UX operating system that is an executable piece of code responsible for managing the computer's resources.

keysh
The command for invoking the Key Shell.

Key Shell
An HP-UX shell which, as an extension of the Korn Shell, uses hierarchical softkey menus and context-sensitive help to aid users in building command lines.

kilobyte
1024 bytes, abbreviated Kbyte.

LAN
Local Area Network.

LP
Line Printer.

login script
The shell script, run at each login, which sets the login environment for your terminal.

special character
A character that has special meaning to the HP-UX shell, as well as to commands such as *find(1)*, and *grep*. The set of metacharacters includes: `*`, `?`, `!`, `[`, `]`, `<`, `>`, `;`, `|`, `'`, `` ` ``, `"`, and `&`. Also called "metacharacters".

node name
The unique identifying name given to a system in a cluster, such as `xyzleslie`.

operating system
The contents of `/hp-ux`, including the kernel, input-output control, system accounting, storage assignment, and other services.
Also see **kernel**.

option
The part of a command line that changes the behavior of a command in certain ways. An **option** is usually preceded by a `-`.

owner
The owner of a file is usually the creator of that file. However, the ownership of a file can be changed by the super-user or the current owner with the *chown(1)* command or the *chown(2)* system call.

PAM
Personal Application Manager.

parent directory
A directory which contains other directories, each of which is then called a **subdirectory** (or **child directory**).

parent process
A pre-existing process which has created a **child process**.

password
An encrypted sequence of characters used by HP-UX to identify an authorized user and to permit authorized login on a system.

path name
A sequence of directory names separated by slashes and ending with a file name.

PID
Process Identification. *Also see* **process identifier**.

process
An program running in machine memory.

process identifier
A unique identification number assigned to all processes by the operating system. Also called **process i.d.**

program (n.)
A unit of executable code, in binary or "source" form. Most HP-UX commands and routines consist of programs.

RAM
Random-access memory.

relative path name
The name of a file, listing all the directories leading to that file in relation to the current working directory.

remote host
A system which is named in your `/etc/hosts`. To log in and perform file transfer tasks on a **remote host** you will need to have a login and a `.rhosts` file on that system. The `.rhosts` file contains your system name or system node name, such as `xyzleslie`. Similarly, to transfer files *from* a

remote system, you will need to have a local `.rhosts` file which contains the system node name of the remote system.

ROM
Read-only memory.

root directory
The highest level directory of the hierarchical file system, from which all other files branch. In HP-UX, the slash (/) character refers to the "root directory".

server
This refers to a computer that provides file access, login access, file transfer, printing and other services across a network to a defined cluster of systems (**cluster nodes**). Same as **cluster server**.

SGID
Set-group identification (permission).

shell
An HP-UX command interpreter (Bourne, Korn, Key or C).

shell script
A file that contains commands that a shell can interpret and run.

shell variable
One of the variables, such as PATH, TERM, SHELL, and HOME that defines your terminal characteristics, home directory, and default search paths, among other things.

standalone
A computer which is not part of a cluster.

standard error
The destination of error and special messages from a program, intended to be used for diagnostic messages. The standard error output is often called `stderr`.

standard input
The source of input data for a program. The standard input file is often called `stdin`.

standard output
The destination of output data from a program. The standard output file is often called `stdout`.

subdirectory
A directory which is located in, or anywhere on a path below, another directory, which is then called its **parent directory**. Sometimes called **child directory**.

subshell
A shell created by another shell to run a program.

SUID
Set-user identification (permission).

superuser
A user with "root permission", usually a system operator or administrator.

system administrator
The person in a group who is assigned to look after the system installation, maintenance, updating, and installing new software for a system or cluster of systems.

system operator
Same as **system administrator**.

trusted computer system
A system that employs sufficient hardware and software integrity measures to allow its use for sensitive or classified information.

TTY
teletypewriter.

UID
User Identification.

user
Any person who interacts directly with a computer system.

username
The name which identifies your account to the `login` program and to the mail systems and other software requiring secure entry.

user ID
An integer which identifies each system user.

working directory
See **current working directory**.

write protect
Using the protective device on your tape cartridge to prevent it from being accidentally overwritten.

Task Index

W

Main Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

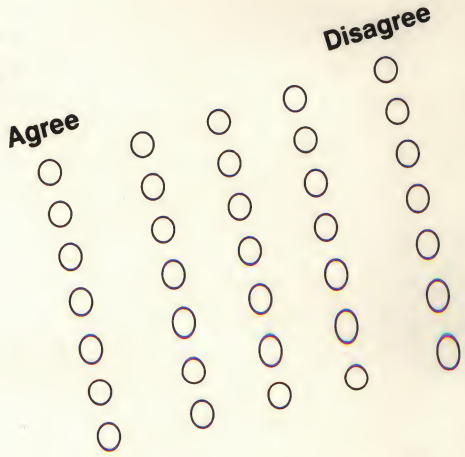
W

Y

Win an HP Calculator!

Your comments and suggestions help us determine how well we meet your needs. **Fill out this card with your name and address enters you into a quarterly drawing for an HP calculator***.

HP PHIGS FORTRAN Binding Reference



The manual is well organized.
It is easy to find information in the manual.
The manual explains features well.
The manual contains enough examples.
The examples are appropriate for my needs.
The manual covers enough topics.
Overall, the manual meets my expectations.

You have used this product:

__ Less than 1 week
__ Less than 1 month
__ Less than 1 year
__ 1 to 2 years
__ More than 2 years

fold

Please write additional comments, particularly if you disagree with a statement above. Use additional pages if you wish. The more specific your comments, the more useful they are to us.

Comments: _____

*Offer expires 12/1/1992. (Manual: B1759-90005 E0191)

Please Tape Here

name and address.

Company: _____

Address: _____

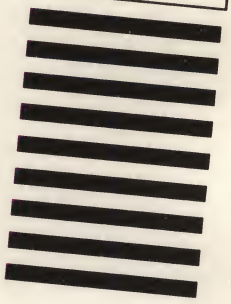
City, State, Zip: _____

Telephone: _____

Additional Comments: _____

Beginner's Guide to HP-UX
Part Number B1862-90000
91

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

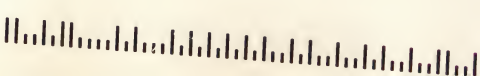


BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Learning Products HP-UX
Hewlett-Packard Company
3404 East Harmony Road
Fort Collins CO 80525-9988



Manufacturin
Part No.
B1862-90000

Manual Part No.
B1862-90000